University of Alberta

Investigating Experience: Temporal Coherence and Empirical Knowledge Representation

by

Anna Koop

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

in

Department of Computing Science

Edmonton, Alberta
Spring 2008

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-45836-5
Our file Notre référence
ISBN: 978-0-494-45836-5

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Experience is the temporal sequence of sensations and actions that are the inputs and outputs of an intelligent agent. This thesis identifies and develops two ideas that arise from treating experience as fundamental to intelligence. *Temporal coherence* is the tendency of experience to be consistent over short periods of time. Results in a simple prediction task and the more complex problem of Computer Go show how an agent with dynamic memory can take advantage of the temporal coherence of its experience to dramatically improve on the performance of the best static solution. *Empirical knowledge representation* is the representation of knowledge that describes and is defined in terms of experience. An empirical knowledge representation that uses the frameworks of predictive representations and options is presented. In a case study of the object concept, this framework is shown capable of representing abstract knowledge in empirical terms.

To my husband Joel,
You make everything possible for me, my love. This is for us.

Acknowledgements

Thanks to my supervisor, Rich Sutton, for his enthusiasm for big ideas and insistence that I have something to contribute to our field. David Silver's hard work, collaboration and engagement with new ideas has made much of this work possible. Thanks to Joel Koop, Elizabeth Wightman and Elliot Ludvig for their input on the text and core ideas.

Many people have provided me with advice and assistance. I owe my mother, Lois Wightman, a lifetime of support and encouragement. Shayna and Michael Bowling have been an inspiration and help throughout my degree. The group at RLAI are some of the best people I have ever worked with, and I especially thank Dan Lizotte and Brian Tanner for timely words and helpful input. Finally, this couldn't have happened without the tremendous support and love of my husband Joel. Thank you.

Table of Contents

1 Investigating Experience 1
- 1.1 Temporal Coherence . . . 1
- 1.2 Empirical Knowledge Representation . . . 3

2 Background for Temporal Coherence 5
- 2.1 Discrete Dynamical Systems . . . 6
- 2.2 Reinforcement Learning . . . 7
- 2.3 Go . . . 10
- 2.4 RLGO . . . 12

3 Tracking and Transience 15
- 3.1 Tracking in the Half-Moon World . . . 16
- 3.2 Transience in the Half-Moon World . . . 20
- 3.3 Simulating Experience in RLGO . . . 22
- 3.4 Long- and Short-Term Memory in RLGO . . . 27
- 3.5 Conclusion . . . 28

4 Learning to Learn 30
- 4.1 Incremental Delta-bar-Delta . . . 31
- 4.2 Step-size Adaptation in the Half-Moon World . . . 33
- 4.3 Temporal Coherence for Task Transfer . . . 37
- 4.4 Conclusion . . . 40

5 Empirical Knowledge Representation 41
- 5.1 Non-Experiential Knowledge Representation . . . 42
- 5.2 Reinforcement Learning . . . 43
- 5.3 Key Components of Empirical Knowledge Representation . . . 44
- 5.4 Options . . . 45
- 5.5 Predictions . . . 47

6 Experiential and Predictive Knowledge Representation 49
- 6.1 Experiential Knowledge Representation . . . 49
- 6.2 Value Function Representation . . . 51
- 6.3 Predictive State Representations . . . 52
- 6.4 Temporal-Difference Networks . . . 54

7 Case Study in Empirical Knowledge Representation: Aspects of Objectness 58
- 7.1 The Beginnings of Existence . . . 59
- 7.2 Persistence . . . 61
- 7.3 Permanence . . . 63
- 7.4 Configural Patterns . . . 65
- 7.5 Predictive Patterns . . . 66
- 7.6 Conclusion . . . 68

8 Conclusion 69
- 8.1 Temporal Coherence . . . 69
- 8.2 Empirical Knowledge Representation . . . 70

Bibliography 72

Chapter 1

Investigating Experience

The view of artificial intelligence (AI) adopted in this work is that AI is about understanding and creating an intelligent agent: a system that responds to the sensations it receives from the larger world by choosing actions that seem to make sense. In such a system, the continual sequence of sensations and actions that are input and output make up the experience of the agent. Experience has received comparatively little attention in artificial intelligence and machine learning research. Emphasis has instead been placed on separating out elements of intelligence such as knowledge representation, planning and learning, and these pieces are frequently investigated outside the context of an agent interacting with an environment. The problem of connecting experience to an underlying reality is too often considered separate from the problem of understanding, planning with and learning about that reality. The view taken in this thesis is that experience is intrinsic to AI. Experience is the data that is always available to the agent—ever-changing but continually being renewed without requiring human intervention or interpretation.

This thesis investigates the benefits of representing knowledge in terms of experience and illustrates how such a representation might encode abstract knowledge.

1.1 Temporal Coherence

Time is an inherent property of experience, but temporal relationships are often ignored in AI research. The typical assumption in machine learning is that there are two separable phases in any AI application: training and testing. All learning occurs during the training phase, and then the agent executes the best learned solution during the testing phase. This division is necessary for purely supervised learning, where the agent must learn from data that has been labeled by a human and then apply that learning to unlabeled data. However, the train and test paradigm dominates even in unsupervised learning or reinforcement learning, where the agent is not constrained by manually labeled data.

An agent that is not restricted to distinct training and testing phases may benefit from the greater flexibility learning from experience can allow. Like the training phase of supervised learning, past

experience provides an agent with data to learn from. Like the unlabeled test data, future experience is the unknown testing ground of an agent's knowledge. However, an agent that is learning from experience has a potential advantage over a supervised learner or any other learner that maintains a sharp division between training and testing. An agent that is continually learning does not have to weight all past training data equally. Such an agent may take advantage of certain temporal characteristics of experience by adapting to the present.

One important temporal characteristic of experience is a tendency for sensations to be consistent over short periods of time. This characteristic is here called *temporal coherence*. Temporally coherent sensations are common in everyday life. The temperature of the room, the colours and brightness of my monitor, the sounds my computer is making and the texture of the keys beneath my fingers—all these tend to be consistent over short periods of time. The temporal coherence of experience can extend beyond immediate sensations as well. My prediction of whether I will win or lose a chess game changes while I am playing, but on a moment-by-moment basis it stays remarkably consistent. What I expect to see when I open my lab door is a temporally coherent prediction. If I have been away for weeks, I expect the change to be more drastic than if I have stepped out for a moment. Ordinary experience is frequently temporally coherent.

When experience is temporally coherent, what an agent is experiencing *now* is a good predictor of near future experience. An agent that is continually learning and adapting to its environment is able to take advantage of the temporal coherence of experience. One way it may do so is by having memory that adapts to recent experience, giving weight to present and near-past experience rather than remembering only the long term average over all experience. Such *dynamic* memory is not necessarily available to learning agents with strictly enforced separation between training and testing phases. The following three chapters investigate temporal coherence and dynamic memory in greater detail.

Chapter 2 presents the background for this work on temporal coherence, introducing discrete dynamical systems, reinforcement learning and the Computer Go testbed. The concept of discrete dynamical systems gives a mathematical framework for understanding intelligent systems that is used throughout this thesis. Reinforcement learning is a framework for learning through interaction that has been successfully applied in psychology and machine learning to understand and direct the behaviour of intelligent agents. Reinforcement learning has much in common with the approach to AI advocated in this thesis, and the Computer Go experiments use the reinforcement learning framework. The rules of Go are described in Chapter 2, along with an introduction to RLGO, the reinforcement-learning program developed by David Silver and collaborators (Silver et al., 2007).

Two approaches to dynamic memory, tracking and transience, are introduced in Chapter 3. Tracking, or forgetting through interference, allows the memory of an agent to change in response to recent experience, replacing past knowledge quickly. Transience, or spontaneous forgetting, allows for the effect of experience to fade as the temporal distance increases. For sensations with high

temporal coherence, these techniques can significantly improve the accuracy of an agent's memory. The experiments with dynamic memory lead to some surprising conclusions: forgetting can be good for knowledge representation and focusing learning on the now can lead to dramatic improvements in performance.

Chapter 4 investigates the implications of temporal coherence for meta-learning. Meta-learning means learning not only the parameters of a knowledge representation but also some of the components that define the learning process. In particular, a parameter that controls the learning rate, and thus the amount of tracking, can be tuned through interaction with an environment. One algorithm for tuning this parameter is applied in both a temporally coherent and temporally incoherent world. In both cases the agent learns the best parameter value, but the improvement from meta-learning dominates only in the temporally coherent world. The difference in the strength of the effect suggests that temporally coherent worlds provide a useful testbed for meta-learning.

1.2 Empirical Knowledge Representation

A knowledge representation is most generally the framework an intelligent agent uses to store and process information. In the field of AI, knowledge representation has come to refer almost exclusively to formal logic systems, where knowledge is captured in a database of statements about symbols and a set of rules for manipulating those statements. While the formal logic approach has had some success, for example in expert systems and applications in computer game-playing, it has some drawbacks. Most notably, it poses difficulties for autonomous agents. In traditional knowledge representation research, knowledge is abstract and symbolic by design and as a result generally requires human maintenance. For autonomous AI, it is desirable for a knowledge representation to be independently meaningful. One way this independence can be achieved is if the knowledge is constructed out of data that is immediately accessible to the agent, such as its experience. When knowledge is represented in a way that is accessible to the agent without human interpretation, it is possible for knowledge to be verified, tuned and even learned autonomously.

Chapters 5 to 7 address the development of a knowledge representation framework that is explicitly based on experience, predictions and time. The underlying data of this knowledge representation is the agent's continual sequence of sensations and actions. All knowledge in this framework is explicitly verifiable against future experience. This general approach is referred to in this work as *empirical knowledge representation*. The first formal framework for empirical knowledge representation is introduced in Chapter 5.

An empirical knowledge representation is a knowledge representation that is grounded in experience and makes predictions that can be verified against experience. Empirical is used here in the sense of empirical science: what is known must be ultimately verified through experimentation. It is not necessary that everything be learned from experience, but all knowledge in this framework must be verifiable through interactions with the environment.

Concepts in the knowledge representation presented in Chapter 5 are grounded in experience using the frameworks of options and predictive representations. These frameworks have been developed by the reinforcement learning research community. The options framework allows for temporal abstraction by providing a principled framework for extending single actions to patterns of behaviour. The framework of predictive representations allows for the abstraction of sensations by formally defining predictions as statements about future experience, computed from past experience. Section 5.4 and Section 5.5 explain these frameworks with the slight adaptations necessary for general knowledge representation.

Chapter 6 provides a brief survey of previous work in experience-oriented artificial intelligence and current developments in the area of predictive representations. Predictive representations have been used to represent several kinds of knowledge, although knowledge representation has not been thought of as the primary motivation for work on predictive representation. Relevant results in predictive representation research are briefly described, then compared and contrasted with the aims of empirical knowledge representation.

A common complaint leveled at grounded representations is that the emphasis on concrete experience makes it difficult, if not impossible, to represent abstract knowledge. The beginning of an answer to this complaint is presented in Chapter 7. Taking the notion of objects as a case study, a series of examples are presented to describe how some of what is meant by the idea of objects can be understood in terms of patterns of experience and represented by the empirical knowledge representation presented in Chapter 5. The investigation suggests that even abstract notions such as existence and permanence may be possible to understand in terms of experience.

Chapter 2

Background for Temporal Coherence

The first results presented in this thesis look specifically at how an agent that learns from experience can benefit from the temporal coherence of its experience. Temporal coherence is a general term for the consistency of data across time. An agent with dynamic memory can take advantage of this characteristic of experience to improve performance. The following chapters use two environment to illustrate this potential advantage: a simple prediction task known as the Half-Moon World and the more difficult problem of Computer Go. This chapter provides a brief introduction to important background concepts for understanding temporal coherence and the experiments of Chapter 3 and Chapter 4.

In this thesis, temporal coherence is studied within the framework of dynamical systems and reinforcement learning. Dynamical systems, presented in Section 2.1, are formal mathematical models for systems that change over time. The experiments presented in this thesis are, more specifically, discrete dynamical systems. In a discrete dynamical system the actions and sensations experienced by an agent system are chosen from a discrete set and alternate over small, fixed-length periods of time known as *timesteps*. When presented as a reinforcement learning problem, the sensations of an underlying discrete dynamical system include a scalar signal known as *reward*. The agent in a reinforcement learning problem must behave in a way that maximizes the reward sensation in some way. The reinforcement learning framework is described in more detail in Section 2.2.

A learning agent that is able to take advantage of the temporal coherence of sensations and predictions is presented in Chapter 3 and tested in the domain of Computer Go. This ancient board game has remained a grand challenge for AI, resisting brute-force solutions due to the huge state space and large branching factor. Go has proved to be problematic for expert systems due in part to the intuitive nature of advanced Go knowledge. The game of Go is introduced in Section 2.3. The experiments presented here treat Go as a reinforcement learning problem, adapting RLGO, the reinforcement-learning Go agent developed by David Silver (Silver et al., 2007). RLGO is described in Section 2.4.

2.1 Discrete Dynamical Systems

A dynamical system is any system that changes over time. One mathematical description of a dynamical system consists of the set $\mathcal{X}$ and the function T. The set $\mathcal{X}$ is the set of all possible states the system might be in.[1] In a discrete dynamical system, the members of $\mathcal{X}$ can be represented with discrete values and the system changes over discrete intervals of time, usually small. These intervals of time are known as *timesteps*. At any given timestep t the current state of the system is denoted by x_t. The function T determines the movement of the system through the state space over time: $T(x_t, x_{t+1})$ returns the probability that when the system is in state x_t on timestep t, the system will be in state x_{t+1} on timestep $t+1$.

In the description of the transition function given above, the probability distribution over future states depends only on the current state, x_t. Because the probability distribution over future states does not depend on the entire history of states, $x_{0:t-1}$, the *Markov property* holds for this system. The Markov property states that the probability of future states depends only on the current state; the probability distribution is conditionally independent of the history, given the current state (Higgins and Keller-McNulty, 1995).

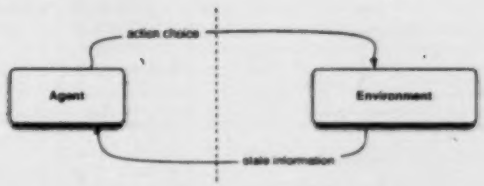


Figure 2.1: The Agent-Environment Interface. The agent takes an action, and in response the environment changes state and sends the agent a sensation. Note that the agent is separate from the environment: it can only affect the state of the world by taking actions, and can only perceive the world through the sensations and reward signals the environment sends back.

A Markov decision problem (MDP) is a dynamical system where the transition function depends not only on the current state but also on an action a_t that is chosen from a set $\mathcal{A}$, and the probability distribution over future states depends only on that pair: $T(x_{t+1}|x_t, a_t)$. MDPs can be described as an agent interacting with an environment, receiving sensations generated by the environment and choosing actions. The agent-environment interface is illustrated in Figure 2.1. The sensation on each timestep, s_t, is generated according to a sensation function from a set $\mathcal{S}$: $S(x_t, s_t)$. In the stochastic, discrete case, $\mathcal{A}$ and $\mathcal{S}$ are discrete sets and $S(x_t, s_t)$ gives the probability of generating sensation s_t when in state x_t. The sensation may directly map to the underlying state. In this fully-observable case, the sensation is known as a *sufficient statistic*, as knowing the sensation at time t is sufficient to determine the probabilities for every possible next sensation. When the sensation is not a sufficient

[1] Note that calligraphic capital letters denote a set and lowercase letters denote members of a set. This convention is used throughout this thesis.

statistic, the system is called a partially observable Markov decision problem.

2.2 Reinforcement Learning

Reinforcement learning is a computational approach to learning from interaction (Sutton and Barto, 1998). The field of reinforcement learning deals with the problem of an agent learning to act in an environment. The agent must choose actions which maximize a scalar reward signal, receiving sensations from the environment at each timestep. It is a special case of a discrete dynamical system, where part of the sensation received by the agent is a numeric reward signal. The reinforcement learning interface is illustrated in Figure 2.2.

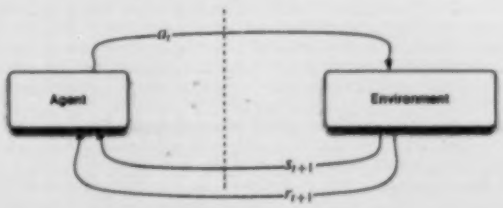


Figure 2.2: The Reinforcement-Learning Interface in a discrete dynamical system. Note the addition of the reward signal.

Reward, which can be thought of as analogous to pleasure and pain, is determined solely by the environment. Because an RL agent learns to maximize reward, the reward function ultimately guides the agent's behaviour and in some sense defines the goal of the system. For example, formulated as a reinforcement learning problem, chess may have a reward signal of +1 if the agent wins the game and 0 if the agent loses. Control of a chemical process in a factory may have a negative reward signal when the temperature approaches a dangerous level, and a positive reward signal related to the speed of the reaction. A maze, formulated as a reinforcement learning problem, might have a reward signal of -1 on each timestep: to maximize its reward, the agent must minimize the number of steps to the goal state. The reward function determines what reward signal will be output to the agent. The reward might be a probabilistic value, and generally depends on the current state and the action chosen by the agent: $\mathcal{R}(r_{t+1}|x_t, a_t)$.

The goal of an RL agent is to maximize reward, but this objective can be measured in many different ways: Should it choose an action that has high reward but leads to states with low reward or choose an action with low immediate reward that leads to states with higher reward? Should it be penalized for selecting an action that normally gives high reward but, because of the probabilistic reward function, happened to return a low value? The question of what exactly to maximize is answered by the *return* function. The return, R_t, is some function of reward values from time t onward (Sutton and Barto, 1998).

Return can be calculated in several ways. The simplest way is to simply add up all the rewards

received from time t onward:

$$R_t = r_t + r_{t+1} + r_{t+2} + ... \tag{2.1}$$

In environments with well-defined endpoints such as games that may be won or lost, or factory control of processes that shut down for the night, return can be meaningfully computed as a simple sum. In continuing tasks, such as a factory processes that is never intentionally turned off or a robot that interacts with its environment in a complex way over a long time span, a different formulation is needed for return. A continuing task has no well-defined end point. If the rewards are summed up from time t into the infinite future, the return could be infinite (either positive or negative, depending on the reward function) regardless of the behaviour of the agent. In continuing tasks, the most common function for return is to sum the discounted future reward values:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \tag{2.2}$$

The discount factor, γ, is a number between 0 and 1 that controls how much weight future reward is given. If γ is very small, the weight given to future reward decreases quickly, so the return is myopic and includes only the reward the agent will receive in the near future. If γ is 0, the return includes only the immediate reward. If γ is high, reward in the distant future is still given significant weight, and the return thus favours long-term performance.

The agent uses its *policy* to chose actions. A deterministic policy function returns the action that the agent chooses in every state. A stochastic policy function returns the probability that the specified action will be chosen from the specified state. For an MDP that is fully observable—that is, where the sensation is a sufficient statistic for the environment state—the policy can be represented as a function that takes the current sensation and an action as input and 1 when the action should be taken and 0 otherwise. The policy may be learned directly from environment interactions, but more often the value function is used as an intermediate structure.

The value function represents how good it is to be in a given state. The value of a state is More formally, the value function is the function $V^\pi(x)$ that maps states to the expected return, given the policy π is followed:

$$V^\pi(x) = E_\pi[R_t | x_t = x]. \tag{2.3}$$

When the transition function and reward function for the environment are known, the value function can be computed in a recursive fashion, using the Bellman equations[2]:

$$V^\pi(x) = E_\pi[r_{t+1} + \gamma V^\pi(x_{t+1})]. \tag{2.4}$$

Computing the exact value of each state can be an expensive process, prohibitive when the state space is large and not possible when the transition and reward functions are unknown (Sutton and Barto, 1998).

[2]The equation for discounted return is given here. The equation generalizes to the simple sum when $\gamma = 1$

An approximate value function, $\hat{V}(x)$, can be learned through temporal-difference (TD) learning. TD learning is often referred to as boot-strapping or 'learning a guess from a guess' (Sutton and Barto, 1998). The TD-learning rule uses the value function estimate of the current state to improve the estimate for the previous state. On each timestep, the agent receives a new reward value, r_{t+1}, and new state information, x_{t+1}. The sum (discounted or not) of the reward actually received and the estimate of the next state's value is like a more accurate estimate of the value of x_t.[3] The difference between this sum and the previous estimate for state x_t is known as δ_t, the TD error at time t:

$$\delta_t = r_{t+1} + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t). \tag{2.5}$$

The TD-learning rule uses the TD error in an incremental update rule:

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha \delta_t. \tag{2.6}$$

The estimate of the previous state's value is changed by some amount controlled by the step-size parameter α and the TD error.

The step-size parameter, α, is sometimes also known as the learning rate, because it influences how quickly the agent learns from experience. It is called a *step-size* parameter because it controls how much the estimated value moves (or "steps") towards the improved estimate. When α is high, the TD error is given more weight and the estimate is changed significantly to reflect the new data. When α is low, more weight is given to the previous estimate and the agent resists changing its value function. If the reward the agent receives in each state has high variance, having a small α can prevent the estimate from chasing the noise in each reward. With an α that decreases at an appropriate rate, TD learning will learn an accurate value function in an environment that satisfied certain properties (Tsitsiklis and van Roy, 1997).

The value function can be used by the agent to calculate a policy, as the agent can use the value function to compare states. The best action is the one that leads to the state with the highest estimated value. Two important policies based on the best action estimate are *greedy* policies and *ϵ-greedy* policies. An agent following a greedy policy always chooses the action that appears to lead to the best state. An ϵ-greedy policy defines a small number between 0 and 1, though usually $\epsilon << 0.1$, that represents the probability of not taking the best action. An agent following an ϵ-greedy policy chooses the best action most of the time, but some of the time (with probability ϵ) it chooses a random action. Taking random actions with some small probability allows for exploration of the state space. If only the best actions are taken, the agent can suffer from an inaccurate value function or simply never learn about large portions of the state space.

The value-function equations described so far are for the table-lookup case, when a single and separate value is learned for each possible state. Such a perfect representation is not always possible, practical or desirable. Table-lookup is impossible when the state-space is larger than memory

[3] In fact, it is guaranteed to approach perfect accuracy with infinite experience, in certain circumstances (Tsitsiklis and van Roy, 1997).

resources allow. When the number of possible states is small enough to fit in memory, table-lookup might still be inappropriate if the sensation received on each timestep is not a sufficient statistic, if the sensation does not map directly to the state of the MDP. Then the agent might need to construct an internal summary of its history that is a sufficient statistic. In either case, if it were possible to represent the value for each state in memory, it might not be practical or desirable to learn those values independently. Simple table-lookup does not allow for generalization between states. States may have common characteristics that an agent can exploit if its value function representation allows information to be shared across similar states. When such generalization is allowed, learning time can be improved, particularly with incremental learning rules, learning rules that update with each new bit of data such as the TD update presented above. When the exact details of every state of the environment are smoothed away, knowledge learned in one state may be applied in another. The speed of learning can be improved when learning is applied to groups of states that are grouped based on some shared characteristics (Rafols et al., 2005).

The structure used in this thesis for state representation is the state-variable vector $\mathbf{x}_t$. In the table-lookup case, this internal state representation is a bit vector identifying the sensation returned by the environment. In the most general case, $\mathbf{x}_t$ can be an arbitrary function of the history.

Function approximation is the technique of using a function of the state variables $\mathbf{x}_t$ to represent the value function. One common choice of function is the logistic or squashing function, which combines the state variables with a set of learned weights, $\mathbf{w}_t$, and scales the result between 0 and 1 with the sigmoid function σ:

$$\hat{V}(\mathbf{x}_t) = \sigma(\mathbf{x}_t) = \frac{1}{1 + e^{-\mathbf{w}_t{}^T \mathbf{x}_t}} \tag{2.7}$$

The learned weights $\mathbf{w}_t$ are multiplied linearly with the state variables, then put through the logistic function σ. The linear portion, $\mathbf{w}_t{}^T \mathbf{x}_t$, allows for fast computations even with a large number of state variables. The logistic function can improve stability during the learning process.

2.3 Go

The game of Go is a challenging domain for artificial intelligence and is used in this thesis to test how agents can exploit temporal coherence. Go is one of the oldest board games in the world. It has also been one of the hardest for computers to master. Only in recent months has any Computer Go program beaten a professional human, and that was one game out of three on a teaching-size board (Silver, 2007). The achievement is not to be belittled, however. Go has long been considered a "grand challenge" problem for AI (Müller, 2002), with the popular notion being that Go is one area humans will always dominate (Macintyre, 2007). The CGOS game server is one popular testing ground for Computer Go players (Dailey, 2007).

The game of Go has deceptively simple rules. The board is typically a 19×19 square, though boards of 13×13, 9×9 and 5×5 are also commonly used. A Go board in mid-play is illustrated

in Figure 2.3. Two players alternate turns, one placing black stones and the other white on the intersections of the board. The sets of adjacent stones of the same colour are known as *blocks*. The open intersections adjacent to a block are the block's *liberties*. A stone is captured and removed from the board when its last liberty is taken by the other player—that is, when the last open intersection adjacent to the block is claimed by the other colour. Blocks might be alive, dead or unsettled. A block is *alive* when, because of the nature of its liberties and position on the board, it can not be captured. One reason a block might be alive is because it has two or more *eyes*. Eyes are open intersections where the other colour cannot play without being immediately captured.[4] A block is *dead* when it can be captured by the opponent regardless of any defensive moves. A block is *unsettled* if it can be saved by a defensive move or captured if the opponent is to play next.

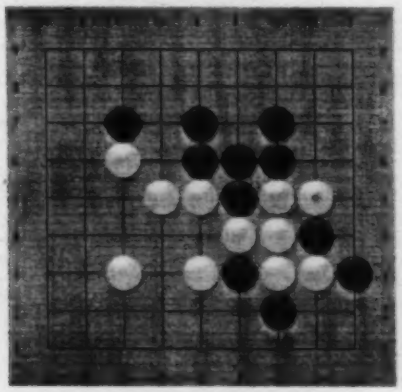


Figure 2.3: A Go board in mid-game.

The player with the most territory at the end of the game wins. Territory is defined as either the total number of captured stones and intersections surrounded by the player or the total number of stones and surrounded intersections. The Elo rating of a Go player, similar to the Elo ratings used in chess, gives an indication of the player's strength relative to other players. An Elo rating of under 2000 roughly indicates a student, or *kyu*, level of play, where a beginner might have an Elo rating of 200. An Elo rating over 2000 roughly indicates a master, or *dan*, level of play. A professional player may have an Elo rating over 2700 (van der Steen, 2007).

One of the reasons Go has been particularly difficult for computers is the large branching factor. In the full 19×19 game, the number of legal states is larger than the estimated number of atoms in the universe, and the branching factor is extremely high (Müller, 2002). However, the branching factor is not the only or even the most significant difficulty. In 9×9 Go, the branching factor is comparable to chess, yet computers cannot play 9×9 Go as well as their chess-playing counterparts. In chess, even programs that do not use special hardware routinely beat experts (Müller, 2002). In contrast, until recently the best Go programs in the world had never beaten a professional player on 9×9 Go.

[4] Some rules do not allow a player to place a stone on an intersection that has no liberties—i.e., commit suicide.

The highest barrier to computer excellence in Go seems to be the difficulty of making a useful evaluation function, caused largely by the non-local interactions between pieces over the course of the game (Schraudolph et al., 2000; Müller, 2002). Human Go players are able to glance at a board and make reasonably accurate guesses about which player is ahead, what groups are in danger, whether stones are alive or dead, and where the attack may be pressed most advantageously. Humans view the Go board simultaneously as a whole and as separable groups, often recognizing patterns that will be played out over dozens of moves. This skill in visual and predictive pattern matching has not yet been mastered by AI.

The reliance of professional Go players on intuition has posed problems for expert systems attempting to emulate them. The expert system approach to computer game playing requires that human experts be able to articulate how they know one position is strong and another weak. As it turns out, humans do not always know why they make the decisions they do (see Polanyi, 1958, for example), and relying on experts to determine what they are basing decisions on can lead programs astray. Because of the difficulty of getting expert humans to verbalize their process accurately, there has been steady interest in using machine learning to create a Go program that can learn to play well directly from games. An approach that uses the reinforcement learning framework, RLGO, is described in the next section. Another learning approach, which has proved so successful almost all the strongest Go programs now use it, is the Upper Confidence Tree (UCT) approach. The UCT approach uses a value function that estimates the probability of winning from each state. A UCT agent selects actions according to a combination of which leads to the highest value state and how uncertain the estimate of the value function is (Gelly and Silver, 2007). In other words, an action with an intermediate value that has not been tried very often could be chosen over an action with a pretty good value that has been tried very often. By combining an estimate of the value with knowledge about how uncertain the estimate is, and using the combination to select actions, UCT balances exploration of the moves with exploitation of the agent's knowledge in a more principled way than the the ϵ-greedy policy described earlier.

2.4 RLGO

The RLGO agent is a reinforcement learning Go player, invented and developed by David Silver and collaborators (Silver et al., 2007). The RLGO agent uses a simple reward function with a reward of 1 for winning and 0 for losing. The agent uses binary features computed from the board position to learn a logistic value function. It learns the parameters of the value function by playing many thousands of games against itself or other players. RLGO is the strongest Go player that does not use UCT currently running on the CGOS 9×9 server.

The state variables used by RLGO are shape-variable functions that return 0 or 1 according to whether the current board position matches their settings, something like visual receptive fields for particular patterns. Each variable may be location dependent or location independent. Location-

dependent variables return 1 only when their pattern is located at a specific position on the board. Location-independent variables return 1 when their pattern is visible at any location on the board. Each variable has a size, which indicates the number and arrangement of intersections it is concerned with: 1×1, 1×2, 2×2, up to 3×3. Within that shape, each variable has a pattern that specifies the arrangement of black stones, white stones, empty intersections, and irrelevant or wildcard locations it recognizes. The full set of variables up to the 3×3 shapes can result in several million state variables, most of which return 0. The sparsity of the binary representation, combined with a logistic value-function approximation, allows for fast evaluation of states.

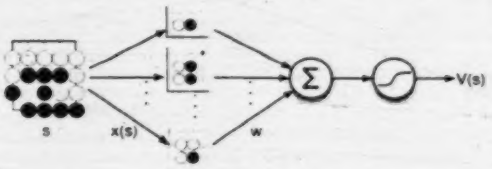


Figure 2.4: Value function construction in RLGO

The value for each state is computed as the logistic of the weighted sum of the state variables:

$$V^{\pi}(s_t) \approx V(\mathbf{x}_t) = \sigma(\mathbf{w}_t^T \mathbf{x}_t). \tag{2.8}$$

The value function computation is illustrated in Figure 2.4. The state variables $\mathbf{x_t}$ are computed directly from the current board position, then combined with the appropriate learned weights and squashed via the logistic function, as described in Section 2.2, so that the value is a number between 0 and 1 that represents the probability of winning the game. During learning, each weight w_t^i can be updated according to the TD learning rule:

$$w_{t+1}^i = w_t^i + \alpha \delta_t x_t^i. \tag{2.9}$$

Although Go is a perfect information game, like most Computer Go players, RLGO does not operate directly on the board position. Rather, it uses the binary features described above. The binary features allow for fast learning: exact board positions are rarely encountered more than once, and the value for every legal board position can be difficult or impossible to represent in memory, depending on the size of the board. The range of sizes and the duplication between location-dependent and -independent shapes allows for information to be shared between similar states. The 3×3 location-dependent variables are the most specific. The 1×1 location-independent variables are the most general and encountered the most often. This combination of specific and general features allows the RLGO agent to learn things that are generally true quickly using the general variables and then learn about specific variations using the more precise variables. The less precise variables provide no more information than the detailed state variables, but can allow for faster, abstract learning and greater transfer to new situations.

RLGO makes further improvements to learning speed by exploiting symmetries in the game of Go: symmetry between players and in the board positions. A valuable shape for white player would be similarly valuable for black, were all the stones reversed. The weight an RLGO agent assigns to a shape when playing black is used with the inverted pattern to evaluate white's position. When RLGO is training through self-play, only one value function needs to be learned and represented. The symmetry of the Go board itself provides another opportunity for weight sharing. State variables that indicate patterns and locations that are invariant under reflection and rotation also share weights.

The game of Go has high temporal coherence: it is rare that the placement of a stone drastically changes the status of a block, particularly to a player that is able to project into the future. The following chapter will explore how an RLGO agent with dynamic memory can benefit from the temporal coherence of the game.

Chapter 3

Tracking and Transience

The ability to build dynamic models of memory is one of the benefits of relating knowledge to experience. An agent that learns directly from the continual sequence of experience may be able to take advantage of certain characteristics of experience, such as temporal coherence, that other models can not. This chapter provides illustrations of how a single best-fit solution over all possible experience may not be as good as knowledge that is customized to the current experience of the agent through dynamic memory. The two ways of adapting memory dynamically that are investigated here are tracking and transience. Tracking occurs when the agent continually updates its internal representation rather than maintaining a single, stationary model. Transience occurs when the passage of time causes the effects of recent experience to fade.

The first set of experiments illustrates tracking, or forgetting by interference. Tracking occurs when new, current information replaces information previously stored in memory. In Section 3.1 a simple prediction task called the Half-Moon World is used to illustrate temporal coherence and the effects of tracking through a high step-size (recall from Section 2.2 that the step-size parameter affects the learning rate). The experiments show that tracking in the Half-Moon World can result in a three-fold improvement in performance over the best stationary memory. The experiments in the Half-Moon World show tracking through a fast learning rate.

The next set of experiments illustrate transience, or spontaneous forgetting. Transience occurs when the passage of time causes the memory of past experience to fade. In Section 3.2 a forgetting term is added to the learning rule of the Half-Moon World agent, so that its memory decays over time whether or not it receives new, relevant observations. The addition of the forgetting term allows further improvements in performance. The degree of improvement relates to how frequently the prediction is verified through experience.

Section 3.3 presents an RLGO agent that tracks through simulated experience. This tracking RLGO agent is pitted against a converging RLGO agent who learns a single, stationary solution, in 5×5 Computer Go. The tracking agent starts with no information and learns during a single game by simulating experience from each state encountered over the game. Even with less overall

learning time than the converging agent, the tracking agent wins the majority of games.[1]

Section 3.4 further investigates the issues of memory by combining long- and short-term memory in RLGO. The long-term memory parameters maintain a stable representation of the long-term benefits of state variables. The short-term parameters use tracking and transience to customize the agent's memory to current experience. The combination of the two types of memory result in a stronger Go player than either of the two independently.

3.1 Tracking in the Half-Moon World

It is rarely possible for any intelligent agent to have a perfect understanding of its world. One reason perfect knowledge can elude an AI agent is its limited memory resources. Often the memory available to the agent is not sufficient to store all of its experience exactly. This section uses a simple environment, the Half-Moon World, to show how tracking can benefit an agent with limited memory resources. The Half-Moon World is a simple prediction task where an agent wanders between two regions of the environment and must predict the colour of the region it is in, black or white. This task is made difficult through strict memory limitations and partial observability.

The Half-Moon World is used here to explore the interaction between the temporal coherence of the environment and the dynamics of the agent's memory. The prediction of black is temporally coherent. If that prediction has recently been verified (*i.e.*, black was observed), then the agent is likely to still be in the black region. The more frequently the agent checks the colour, the higher the temporal coherence of the sensation it observes. As the probability of looking up decreases, the temporal coherence of its sensation decreases, and the sensation becomes almost uniformly random.

An agent in the Half-Moon World has limited memory resources. How those resources are allocated affects the accuracy of its predictions. The agent's memory resources are not sufficient to learn about every state in the world, and so the agent is guaranteed to have some error in its prediction. A converging agent learns the single best solution—the one that, given the memory resources available, minimizes the error or loss equally over all states experienced. A tracking agent learns a dynamic solution—one that adapts to recent experience. Tracking allows an agent to take advantage of the temporal structure in its experience and improve prediction accuracy.

The tracking algorithm

A tracking agent adapts its memory in response to experience. In the following experiments, this adaptation is done with a large step-size in the learning rule. In the TD learning rule presented in Section 2.2, the weight placed on recent experience is controlled through the step-size parameter α. A high α means memory changes significantly to reflect recent experience, whereas a low α means that long-term memory is given precedence, and memory changes only slightly towards the most recent experience.

[1] Results and figures were first published in ICML 2007, (Sutton et al., 2007).

Both the tracking agent and the converging agent use the logistic function with a single memory parameter, w_t, to predict the probability of seeing black, y_t:

$$y_t = \frac{1}{1 + e^{-w_t}}. \tag{3.1}$$

The loss in the prediction y_t is computed on every timestep on which the `look` action is taken, according to the cross-entropy loss between the prediction and the sensation on the following timestep, s_{t+1}:

$$L_t = -s_{t+1} \log(y_t) - (1 - s_{t+1}) \log(1 - y_t). \tag{3.2}$$

The cross-entropy loss is a standard loss measure for the prediction of binary features because it is sensitive to changes in the prediction (Hastie et al., 2001). The cross-entropy loss approaches infinity as the prediction y_t approaches extreme wrong values and quickly approaches 0 as the y_t approaches correct values.

The memory parameter w_t is updated according to a TD-like learning rule:[2]:

$$w_{t+1} = w_t + \alpha(s_{t+1} - y_t). \tag{3.3}$$

The weight update occurs on every timestep the `look` action is taken.

A tracking agent uses a large α value so that its memory parameter w_t is shifted in response to recent experience. A converging agent uses a decreasing α to learn the single best w_t that minimizes cross-entropy loss over all timesteps.

Task

The Half-Moon World is illustrated in Figure 3.1. There are twenty states, half in a black region and half in a white region. The agent can chose between two actions. When the `wander` action is taken, the agent moves to either neighbouring state with equal probability, and the environment returns a sensation of 0. When the `look` action is taken, the agent does not move, and the environment returns a sensation of 1 if the agent is currently in the black region and 0 otherwise. The temporal coherence of the agent's sensation increases when the `look` action is taken more frequently. It decreases the longer the agent repeats the `wander` action without taking the `look` action. By manipulating how frequently the agent takes the `look` action, the interaction between tracking and temporal coherence can be studied in detail.

The agent's task is to predict the probability it will sense 1 if it takes the `look` action. This task would be simple if the environment was a fully-observable MDP (that is, if the agent was able to sense which state it was in rather than only sense a single bit). It would be possible to solve completely if the agent was allowed to build up a large enough representation for a complete state model. However, with the memory limitations described above it is not so simple.

[2]The learning rule is the incremental update for gradient descent in L_t. For a related derivation, see Section 4.1

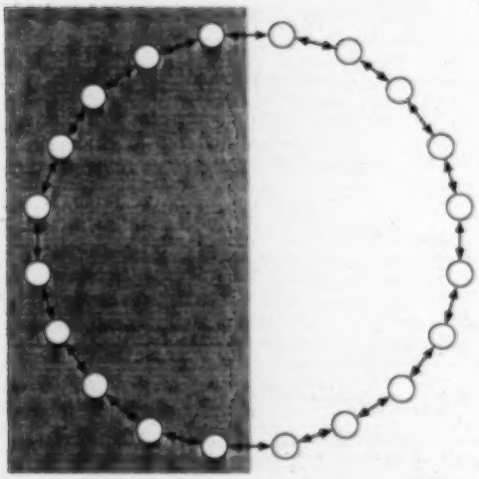


Figure 3.1: The Half-Moon World. The agent follows a uniform random walk, occasionally observing the colour of the region it is in through a binary sensation bit.

A sample trajectory is illustrated in Figure 3.2. The colour of the region the agent is in during a 50 timestep period is shown at the bottom of the figure. The overall probability of seeing black is given by the green dashed line. The solid blue line shows a prediction that tracks the sensation: when the `wander` action is taken, the prediction remains steady as no updates to the memory parameter are made. When the `look` action is taken, the prediction is updated towards the sensation. When the agent is in the black region, as illustrated from time 0 to time 15, the prediction y_t increases and approaches 1. When the agent moves into the white region, the prediction is decreased, approaching 0. The speed of that adaptation is determined by the step-size used in the learning rule.

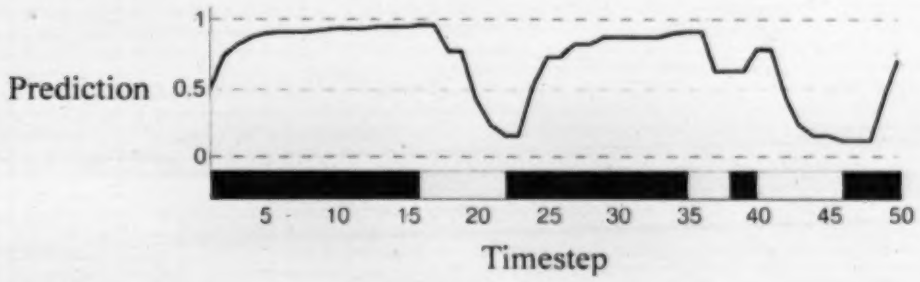


Figure 3.2: A sample trajectory in the Half-Moon World, showing the prediction made by a tracking agent on each timestep and the actual colour of the current region. The prediction is modified only on timesteps on which the colour is observed. Here $\alpha = 2$. Adapting to the current experience can lead to better predictions than focusing on the single best solution.

Experiment details

The following experiments used a twenty-state Half-Moon World with ten contiguous black and ten contiguous white states. The probability of the agent taking the `look` action on each timestep was .8, .5 and .2. The values of α ranged from .003 to 64 in powers of 2.

For each value of α and each `look` probability, the mean loss was measured over 30 episodes of 200,000 steps, counting only the timesteps on which the `look` action was taken. The mean loss per `look` timestep was recorded for the last 100,000 steps, to remove any effect of initial conditions. The mean loss and standard error of the mean loss per episode are shown in Figure 3.3.

Results

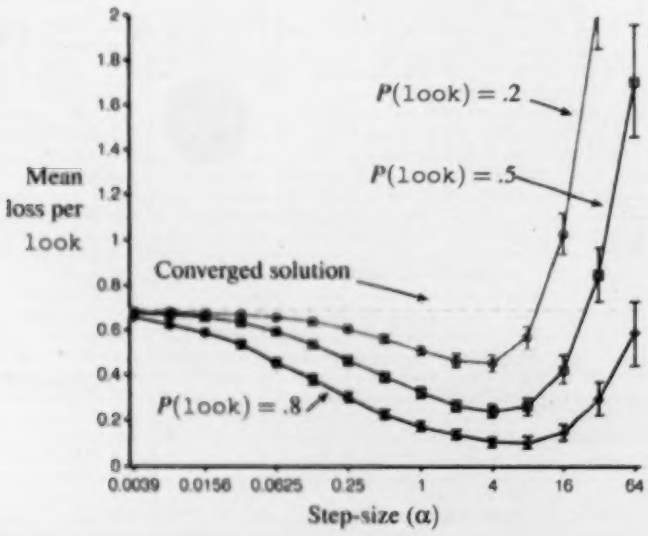


Figure 3.3: Comparison of the mean cross-entropy loss for tracking agents in the Half-Moon World. Loss is measured per timestep where the `look` action was taken. The dotted green line marks the loss of the converged solution. The solid blue lines show the loss for tracking agents with α set according to the x axis. Temporal coherence is varied through manipulation of the `look` probabilities. Standard error bars are given.

The cross-entropy loss per step of the converged solution was .69, illustrated in Figure 3.3 with the green dashed line. For very large values of α, the tracking solutions had higher mean loss than the converged solution. For intermediate values of α, the tracking solutions had significantly lower loss than the converged solution. Which α value was best varied depending on the temporal coherence. When the `look` probability was .8, an α value of 8 was best, with a mean loss of 0.10. When the `look` probability was .5, an α value of 4 was best, with a mean loss of .24. Even with a `look` probability of .2, an α value of 4 improved on the converged solution, with a mean loss of .45. In all cases, as the α values decreased the mean loss approached that of the converged solution.

Discussion

Very large step-sizes result in higher loss and greater variance in mean loss. With a large step-size, it is possible for the agent to become over-confident in its prediction, driving the memory parameter high enough (or low enough) that when the agent moves into a different region it takes many steps before the learning rule's corrections to the parameter are sufficient to change its prediction. The large number of steps needed to change the memory parameter, even when the error is high, is partly an effect of the logistic function. The prediction is bounded by 0 and 1, but as the prediction approaches extreme values, it takes much larger changes in the parameter value to have a noticeable effect on the prediction. Thus, even with a large step-size, when the prediction has been driven to an extreme value it cannot change back quickly.

Very small step-sizes approach the loss of the converged solution, as a decreasing step-size would converge to the single best solution. In the case of a small step-size, changes to the parameter in response to prediction error are so small that the agent is likely to move to a new region before the prediction is driven to extremes. With small α values, the prediction does not move very far from .5, the long-term average.

Tracking through a reasonably large step-size can result in drastic improvements in prediction accuracy. The greater the temporal coherence of the sensation being predicted, the greater the improvement provided by tracking. Tracking allows the agent to take advantage of the structure in its experience without that structure being explicitly programmed in. The best step-size for a tracking agent depends on temporal coherence of the target of its prediction. The next chapter will illustrate how the best step-size can be learned through interaction with the environment.

3.2 Transience in the Half-Moon World

Transient memory, or spontaneous forgetting, is the familiar kind of forgetting where things slip from your memory for no (apparent) reason other than the passage of time. Spontaneous forgetting provides an additional way to represent temporal coherence. The temporal coherence of the Half-Moon World means not only that the most recent sensation has particular relevance, but also that the longer ago the sensation was perceived, the less relevance it has. The particular relevance of the most recent sensation is captured by a tracking parameter that updates towards the most recent observation, illustrated in Section 3.1. In order to allow the sensation's relevance to fade over time, a new parameter, the spontaneous forgetting term, is introduced here.

The transient algorithm

The weight update equation now has two parts: the correction due to error, as before, and a decayed weight. The step-size α, as before, determines how much the knowledge of the agent is updated in response to the current observation. The new decay parameter, ψ, determines how long that effect

holds in the absence of other updates.

The new memory update occurs on every timestep, with the error δ_t defined as 0 on timesteps when the `look` action is not taken and the difference between the sensation and the prediction, $\delta_t = s_{t+1} - y_t$, otherwise:

$$w_{t+1} = \psi w_t + \alpha \delta_t. \tag{3.4}$$

The decay parameter is bounded: $\psi \in (0, 1)$. If ψ is 0, there is no memory from step-to-step. If ψ is 1, then the weight update is the usual update rule as described in Section 3.1.

Experiment details

As before, results are averaged over 30 trials of 200,000 `look` steps in the Half-Moon World. The `look` probabilities were again .8, .5 and .2, and each was tested with several settings of ψ. The step-size α was set to 4, which was found to be the best value overall in the tracking experiments (see Figure 3.3. Again, to remove the effect of initial conditions, only the mean loss for the last 100,000 `look` steps was measured. Results are illustrated in Figure 3.4.

Results

For $\psi = 1$ the learning-rule update is equivalent to the tracking case with $\alpha = 4$, and thus again the greatest improvement was seen the case of highest temporal coherence, where the probability of taking the `look` action was .8. The addition of the decay parameter did not improve the prediction error when the probability of taking the `look` action was high, as the lowest loss in this world was with $\psi = 1$. In the world with intermediate temporal coherence, the lowest loss occurred with $\psi = .9$, though the difference was barely statistically significant. The greatest improvement was in the world with lowest probability of taking the `look` action, where ψ values between .2 and .9 resulted in improved loss. In all cases, the standard error decreased with the size of the decay parameter, with the largest standard error coinciding with $\psi = 1$.

Discussion

When the agent is frequently looking up, the error term δ_t seems to dominate the learning update, and improvements due to the decay parameter are slight. When the agent is looking up infrequently, the decay parameter seems to be particularly important for reducing the effect of tracking over time. These results suggest that the best ψ depends not only on the temporal coherence of the feature being predicted, but also on the frequency of verification. If the agent frequently verifies its prediction against real experience, then correcting towards that experience provides the most benefit. If the agent does not frequently check, then correcting its prediction towards that experience is useful, but it is also beneficial to shift the prediction towards the long-term average.

The decay parameter ψ is similar to that used for weight decay in neural networks (Krogh and Hertz, 1992), a machine-learning technique for learning arbitrary functions, which in turn is related

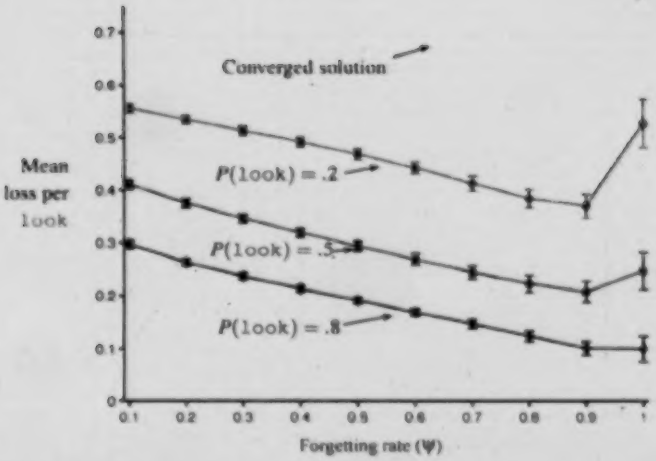


Figure 3.4: Comparison of the mean cross-entropy loss for transient agents in the Half-Moon World. Loss is measured per timestep where the `look` action was taken. The dotted line marks the loss of the converged solution. The solid lines show the loss for transient agents with ψ set according to the x axis. Temporal coherence is varied through manipulation of the `look` probabilities. Standard error bars are given.

to ridge regression (Orr, 1996). Ridge regression helps in ill-posed problems (where there is more than one solution) by adding the constraint that the weights be small. This constraint increases stability. The increased stability can be seen in the results above, where having a high decay parameter decreased the standard error of the mean.

The fact that the long-term best solution is $w_i = 0$ provided an extra benefit in the results above. The decay parameter ψ^i moves the weight on the ith state variable towards 0 over time. This shift towards 0 is generally appropriate in that it causes the importance of that state variable to decay. In the experiment presented here, it has the secondary effect of moving the weight towards the converged result. Thus, when verified through experience, the prediction jumped toward a fixed value. When experience was not available, the prediction shifted towards the long-term average. The shift towards the long-term average can be captured in general by maintaining a long- and short-term set of parameters. The short-term parameters may track and decay quickly, while the long-term parameters maintain the best converged solution. The combination of long- and short-term memory parameters is explored in Section 3.4.

3.3 Simulating Experience in RLGO

The experiments in the Half-Moon World show that it is possible for a dynamic solution to perform significantly better than the best converged solution, when there are limited memory resources and temporal coherence in the relevant sensations. Both these conditions also hold for the game of Go. It

is impossible to exactly represent all possible games and game positions on any existing computer, so function approximation is necessary. Furthermore, the probability of winning has strong temporal coherence during a game: the placement of a single stone can sometimes drastically change the course of the game, but usually does not. The following experiment uses the RLGO framework described in Section 2.4 to compare a tracking agent to a converging agent. The converging agent uses a typical approach to learning in computer games—it learns the best overall policy or value function estimate through extensive offline play and then uses that learned solution during games.

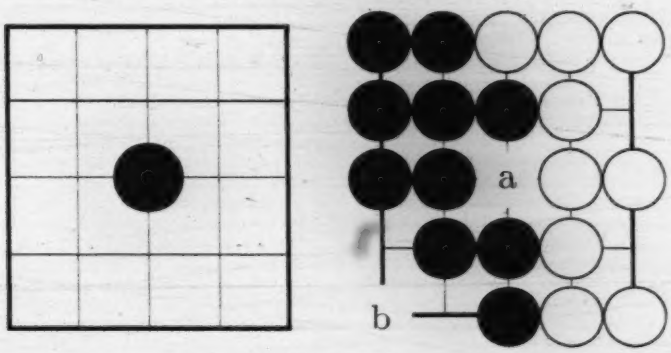


Figure 3.5: (Left) A 1×1, location-dependent feature with a central black stone. (Right) With Black to play, move b is the winning move. Playing at b creates two eyes in the black block, making that block alive (impossible for White to capture). If Black plays at a, then White can play at b, and the black block is dead (impossible for Black to save). Using 1×1 features, the converging agent plays centrally at a, having learned a high weight for the feature that has a black stone there. However, the tracking agent learns that Black must play at b in this particular game, even though in general a corner move is not best.

An RLGO agent's memory is stored in the weights it uses to compute the value function (see Section 2.4). The weights an RLGO agent learns for each binary state variable represent how that particular feature of the board contributes to the probability of winning the game. A converging agent learns those weights over the course of many games, with a small or decreasing step-size. The end result for a converging agent is an estimate of the worth of each state variable over all games. A tracking agent adjusts its value function to the positions of the current game: in the algorithm presented below, it adjusts its value function by using simulated experience. The end result for a tracking agent is an estimate of the worth of each feature for the current game. Figure 3.3 illustrates the difference between a tracking agent's estimate and a converging agent's estimate for a location-dependent 1×1 feature. This feature matches a black stone in the centre of the board. The converging agent learns a high weight for this feature—meaning that in general, playing at a is a good move, as indeed it is. The board position for a particular 5×5 game is illustrated on the right. If Black plays at a, the generally good central position, White is free to play at b, rendering the black block dead. If Black plays at b, a generally poor corner position, it creates two eyes in the black block, guaranteeing the block is alive and winning the game. In this example, knowing what is best

now means winning the game and relying on what is usually best means losing.

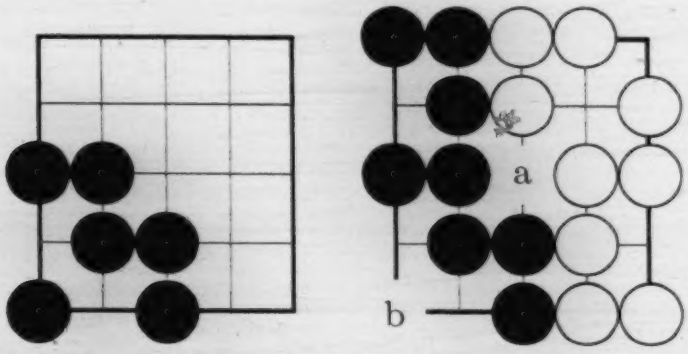


Figure 3.6: (Left) A 3×3 feature making two eyes in the corner. (Right) With Black to play, move a is now the winning move. Using 3×3 features, the converging agent makes two eyes at b, knowing this to be a valuable shape in general. However, the tracking agent can determine that move b is redundant (Black already has two eyes) and learns to play the winning move at a.

State variables that take account of larger portions of the board are more able to capture specific situations and complex stone interactions, but unless the state variables are detailed enough to distinguish between all board positions it can be possible that the particulars of the current game differ drastically from what is generally best. Another dilemma using the 3×3 features is illustrated in Figure 3.3. The figure on the left shows a location-dependent 3×3 feature that makes two eyes in the corner. This shape feature is generally a very strong position, as a block that contains two eyes pattern is alive. It is not a good play in the game illustrated on the right. The black block already has two eyes, beyond the boundaries of the 3×3 feature. Playing at b is unnecessary and gives up the territory at a to the white player, in turn giving up the game. On the full-size 19×19 board, the complex interactions between the stones all but guarantee situations where the representation is not quite detailed enough to include crucial knowledge.

The tracking algorithm

A tracking RLGO agent tracks the current game by customizing its value function through simulated experience. On every turn, before choosing a move through greedy action selection, it simulates some large number of games using self-play starting from the current board position. Because of the sparse, binary state variables, the tracking agent is able to simulate many games within the time allowed for legal moves.

Task

The game of Go was explained in Section 2.3. In the following experiments, a tracking agent plays 5×5 Go against a converging agent that has learned a good value function through self-play. Both agents use the RLGO framework described in Section 2.4. Each agent plays an even number of

games starting as Black and White, as Black has an advantage in 5×5 Go: with perfect play, Black wins (van der Werf et al., 2003).

Recall that RLGO agents both use a logistic value function over binary state variables, and the output of the value function the represents the probability that the agent will win the game. The weights for each state variable x_t^i are learned according to the TD learning rule:

$$w_{t+1}^i = w_t^i + \alpha \delta_t x_t^i. \tag{3.5}$$

The learning rule uses TD error, $\delta_t = (r_{t+1} + V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t))$, introduced in Section 2.2. The TD error is multiplied with the step-size α and the value of the state variable at time t, x_t. Both agents use a greedy policy during play and an ϵ-greedy policy to train through self-play.

Experiment details

The following experiments compared a converging RLGO agent to a tracking RLGO agent in 5×5 Computer Go with state variables with three levels of specificity. The first experiment used only the 1×1 state variables. The second included increasingly complex state variables up to 2×2, and the third included all state variables up to 3×3. The weights were initialized to small random values. A total of 200 games were played, with each agent starting an even number of games as black.

The converging agent was trained offline with 250,000 self-play games for each game against the tracking agent. Weights were randomly initialized each time to protect the converging agent from settling into a local optima—finding a solution that seems best given the starting point, but in fact could be better. The step-size $\alpha = \frac{0.1}{||\mathbf{x}_t||}$, which is .1 divided by the number of active state variables. This α was the best step-size found through hand-tuning (Sutton et al., 2007).

The tracking agent simulated 10,000 self-play games on every turn. Because 5×5 Go is typically decided within the first 25 moves, the tracking agent usually received less training overall than the converging agent and never received more.

The percentage wins and total CPU time for each agent are given in Table 3.1 and Table 3.2 respectively. The percentage wins as each colour is given in Figure 3.7.

Results

For all state-variable sets, the tracking agent won the majority of games while taking less than half the CPU time to train. Increasing the specificity of the state variables resulted in more wins for the tracking agent, from 82% wins with the 1×1 state variables to 93% wins with the 3×3 state variables.

Discussion

The tracking agent beat the converging agent even when both were using a more informative representation. This result is slightly surprising, given that memory limitations were put forward in Section 3.1 as one of the reasons tracking is particularly important. The strength of the tracking agent

Features	Tracking beats converging		
	Black	White	Total
1×1	82%	43%	62.5%
2×2	90%	71%	80.5%
3×3	93%	80%	86.5%

Table 3.1: Percentage of 5×5 Go games won by the tracking agent playing against the converging agent when playing as Black (first to move) and as White.

Features	Total variables	CPU (minutes)	
		Tracking	Converging
1×1	75	3.5	10.1
2×2	1371	5.7	13.8
3×3	178518	9.1	22.2

Table 3.2: Memory and CPU requirements for tracking and converging agents. The total number of binary state variables indicates the memory consumption. The CPU time is the average training time required to play a complete game: 250,000 episodes of training for the converging agent; 10,000 episodes of training per move for the tracking agent.

may be due to an interaction between the representation and the game itself. It is possible for the black player to always win in 5×5 Go. As the agents' memory becomes more detailed, and therefore more expressive, the ability of both agents approaches near-optimal play, and the edge the tracking agent received from tuning its representation is enough to let it continue to beat the converging agent. With only 1×1 state variables, there may not be enough distinctions for even the tracking agent to determine the best moves.

It is also, perhaps, surprising that simulating 10,000 games on every move is at all feasible. As can be seen from Table 3.2, the simulation is quite fast. The tracking RLGO agent is able to play on the CGOS servers in realtime, where there is a time limit per turn. There are also further opportunities for speeding up the algorithm. Simulating far fewer games but starting with a better initial value function might lead to improvements. Some experiments that combine tracking with long-term memory are presented in Section 3.4.

Tracking through simulated experience is particularly applicable to two-player games like Go, where the transition function of the environment is a combination of the agent's own policy and the opponent's policy. The agent's policy provides an approximate model of the opponent's policy and thus the environment, without requiring extra memory or learning time. The model may not be accurate—the opponent could be using any policy—but it is available without dedicating extra resources to the problem of modeling the environment. In more complicated games or other environments, a predictive model needs to be learned separately, but can then be used to simulate experience for tracking. It may prove that a combination of learning from direct experience and simulating experience is ideal for an intelligent agent.

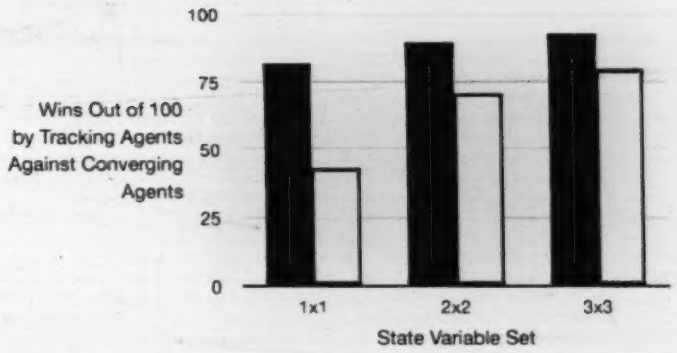


Figure 3.7: Wins by tracking agents against converging agents in 5×5 Go.

3.4 Long- and Short-Term Memory in RLGO

The RLGO agent presented in Section 3.3 has no long-term memory: the value function is re-initialized to small random values at the beginning of each game. Although tracking clearly provides a benefit, long-term best solutions can also be useful. David Silver's latest RLGO program, RLGO 2.1, uses a combination of long- and short-term memory parameters. His results are presented below.[3] On the 9×9 CGOS servers, RLGO 2.1 is ranked in the top ten Computer Go players with an Elo rating of 1880.

The RLGO 2.1 agent maintains two sets of weights over the state variables described in Section 2.4. The long-term parameters are learned offline through ϵ-greedy self-play. During games, short-term parameters are also learned. The short-term parameters do not use weight sharing and are learned as described in Section 3.3. The value of each state is a function of these two sets of parameters. In order to select a move, the agent simulates some number of moves, including the opponent's response, and chooses the action that led to the highest-value state in simulation. This technique is known as *sample-based search*. It differs from a typical greedy policy in that the agent considers more than the immediate next state when choosing the action that led to the best value.

Silver ran three RLGO agents against the open-source Go program GnuGo to compare the performance of long- and short-term memory agents. One RLGO agent used only long-term parameters, one only short-term, and one combined long- and short-term memory. The full set of state variables to 3×3 were used. The long-term parameters were learned during 100,000 games of self-play. GnuGo played at level 0, the fastest and weakest level. The short-term parameters were learned over only 1000 simulated games on each move. The step-size was set to $\alpha = \frac{0.1}{||\mathbf{x}_t||}$. Each agent played 200 games, split between black and white as before.

The agent using only long-term memory won 5.0% games against GnuGo. The agent using only short-term memory won 18.0% of games, and the agent using both long- and short-term memory

[3] Personal communication, used with permission.

won 32.0% of games.

RLGO 2.1 was also tested against GnuGo at full strength, level 10. In this case both long- and short-term parameters were used and the short-term parameters learned through 10,000 simulated games on each move. In this case, RLGO won 57% of 200 games against GnuGo. This version, playing 100 games on the 9×9 Computer Go Open Server, reached an Elo rating of 1880, beating the best handcrafted programs and the strongest converging agent.

The tracking RLGO agent illustrates that an agent with all its memory resources devoted to continually learning improves on an agent with all its memory resources devoted to the best-in-general solution. Silver's RLGO 2.1 results illustrate that a combination can be best. The agent that devotes some resources to continually learning and some to learning a long-term, general solution plays significantly better than one using either type of memory in isolation. It appears that the combination of specializing memory to now and learning what is generally true provides the strongest representation and best results.

3.5 Conclusion

The temporal sequence of experience profoundly affects what we learn and how we learn it. Temporal characteristics of experience, such as temporal coherence, are available to an agent with dynamic memory. Continual learning through tracking is one form of dynamic memory that allows the agent to take advantage of the temporal coherence of its experience and adapt to the current situation.

Continual learning systems have been studied by the machine learning and control theory communities, but almost always in the context of dynamic environments. This chapter showed how two stationary environments, the Half-Moon World with its constant environment and the Go games against an opponent with a single, fixed policy, still benefited from continual learning. The benefit of tracking in stationary environments is due to the limited memory resources—the inability of the agents to perfectly represent the stationary environments induces nonstationarity in the knowledge representation. It is perhaps not a surprising result, but one that is frequently brushed over. Learning algorithms are often tested only against their ability to find the single best solution, with strict separation between the training and testing phases. The results above suggest that researchers are losing more than they might realize with a strict focus on stationary, globally optimal solutions. Segmenting and rearranging experience or collecting data for learning offline can be useful for learning long-term, generally applicable solutions. It appears, however, that important information can be lost when temporal structure is ignored.

In this chapter, tracking and continual-learning methods have been set in contrast to converging to a global solution. Tracking can also be thought of as converging quickly to a locally optimal solution. The large step-size in the Half-Moon World allowed quick steps along the gradient of error towards the minimum at that timestep. The simulated experience of the tracking agent in RLGO allowed the value function to converge (given sufficient simulated experience) to the optimal

solution from the current state. In both those cases, convergence itself is not the problem. Allocation of memory resources towards the temporary but critical experience of the moment is what matters. The idea of a single, stationary solution is rejected in favour of a method for finding the optimal solution for *now*. This ability to focus learning and memory resources on what is impermanent may be critical for an autonomous, intelligent agent.

The results presented here suggest that knowledge is best understood as more than the fixed product of learning. Knowledge can be understood as dynamic: constantly being verified against experience and changing in response to it. A continual-learning agent can make use of the recent past, simulated futures and the richness of the present. The learning process is a crucial part of the knowledge representation of the agent.

Chapter 4

Learning to Learn

An intelligent agent can benefit not only from learning the parameters of its knowledge representation through experience, but also from learning how to learn. Section 3.1 showed how significantly a good choice of step-size parameter can improve results. Learning this meta-parameter through interaction with the world can be as useful as learning the regular parameters.

Meta-learning is used here to refer to algorithms that change the process of learning itself. The automatic learning of useful state variables is one example of meta-learning. Adapting step-size is another example. Through meta-learning, the building blocks of the agent's representation can be autonomously adapted according to the agent's experience.

Section 4.1 presents one particular algorithm for adapting step-size, the logistic form of the incremental delta-bar-delta (IDBD) algorithm. The IDBD algorithm provides a way of adapting step-size online in response to experience (Sutton, 1992a). In Section 4.2, IDBD is tested in the Half-Moon World and is shown to provide improved performance over the fixed-α tracking agent.

One of the outstanding problems in meta-learning research is that it can be difficult to prove the benefit provided by meta-learning. Meta-learning takes time and data. On a single task, the benefits of learning the best representation can be outweighed by the benefits of learning quickly with an adequate representation. The natural solution to this dilemma is to propose a sequence of tasks and demonstrate the cumulative benefit of meta-learning (Caruana, 2005). But how should those tasks be chosen? If the tasks are too different, meta-learning does not transfer, and no benefit can be demonstrated. If the tasks are too similar, meta-learning is unnecessary as the solution for one is adequate for the other. The experiments of Section 4.2 show that meta-learning provides a significant improvement over the fixed-α agent even on the single prediction task of the Half-Moon World. Section 4.3 explores why this discrepancy in improved performance might be so, proposing that in temporally coherent environments, a sequence of tasks arises naturally from the temporal dynamics of the environment. In the Half-Moon World, the combination of memory limitations and the resultant infinite sequence of temporally coherent regions provides us with a sequence of tasks that arises naturally out of a single environment. The effects of meta-learning, in this case through logistic IDBD, are shown to be greater on the temporally coherent Half-Moon World than a

similar environment with no temporal coherence. Temporally coherent environments thus provide a potential testbed for meta-learning algorithms.

4.1 Incremental Delta-bar-Delta

Adapting step-size has been investigated in the fields of statistics, control theory and machine learning. In most cases, the goal of step-size adaptation is to converge quickly to the global optimum. The concern of step-size adaptation for tracking is to find the step-size that results in the lowest error for a continual-learning agent. This section extends the work of Sutton's gradient-descent meta-learning algorithm (Sutton, 1992a,b). The linear IDBD algorithm allows step-sizes to be tuned individually for each state variable. It has been used as a method of discovering which state variables are most important to the knowledge representation: when state variables have large step-sizes, quickly adapting their parameters to changes reduces error.

Schraudolph's work deserves special mention here. His stochastic meta-descent framework provides a general model of gradient-descent step-size learning (Schraudolph, 1999). The derivation below is a special case of the stochastic meta-descent framework. In Schraudolph's version of stochastic meta-descent, he uses an approximation to the Hessian where Sutton uses a diagonalization and an approximation of the exponential function where Sutton uses the exact function.

The IDBD algorithm uses gradient descent, as does the fixed-α learning rule, but in the space of a new parameter β^i rather than only in w^i. The update rules are derived here for the most general case, where the prediction is computed as a logistic function of the state-variable vector at time t, $\mathbf{x}_t$, and the current parameter vector, $\mathbf{w}_t$:

$$y_t = \frac{1}{1 + e^{-\mathbf{w}_t{}^T \mathbf{x}_t}}. \tag{4.1}$$

The equation to minimize is again the cross-entropy loss, L_t:

$$L_t = -z_t \log(y_t) - (1 - z_t) \log(1 - y_t). \tag{4.2}$$

The equations here use z_t rather than s_{t+1}, as in Equation 3.2 to indicate that the target value need not be a sensation directly. It may be the TD-target, $r_t + V^\pi(s_{t+1})$, as in the RLGO algorithm presented in Section 2.4, or the sensation on the next timestep, as in the Half-Moon World presented in Section 3.1, or any other function of the agent's representation.

The weight update rule is similar to that for the scalar case used in the Half-Moon World, but with the α now indexed by time:

$$w^i_{t+1} = w^i_t + \alpha^i_{t+1} \delta_t x^i_t. \tag{4.3}$$

The step-size α^i_t for state variable i at time t is calculated as the exponential of β^i:

$$\alpha^i_t = e^{\beta^i_t}. \tag{4.4}$$

The parameter β_t^i should be adjusted to minimize future loss. The adjustment can be done with the gradient-descent rule, taking the derivative with respect to β^i. This derivative can be thought of as the derivative of the loss with respect to an infinitesimal change in β_t^i at all timesteps. Let $h_t^i = \frac{\partial w_t^i}{\partial \beta^i}$, μ be a meta-step-size parameter for the β_t^i update and δ_t be the difference between the prediction and target, $\delta_t = z_t - y_t$. Then:

$$
\begin{aligned}
\beta_{t+1}^i &= \beta_t^i - \mu \frac{\partial L_t}{\partial \beta^i} \\
&= \beta_t^i - \mu \frac{\partial}{\partial \beta^i} [-z_t \log(y_t) - (1 - z_t) \log(1 - y_t)] \\
&= \beta_t^i + \mu \frac{\partial z_t \log(y_t)}{\partial \beta^i} + \mu \frac{\partial (1 - z_t) \log(1 - y_t)}{\partial \beta^i} \\
&= \beta_t^i + \mu z_t \frac{1}{y_t} \frac{\partial y_t}{\partial \beta^i} - \mu (1 - z_t) \frac{1}{1 - y_t} \frac{\partial y_t}{\partial \beta^i} \\
&= \beta_t^i + \mu z_t \frac{1}{y_t} y_t (1 - y_t) \frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i} - \mu (1 - z_t) \frac{1}{1 - y_t} y_t (1 - y_t) \frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i} \\
&= \beta_t^i + \mu z_t (1 - y_t) \frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i} - \mu (1 - z_t) y_t \frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i} \\
&= \beta_t^i + \mu z_t (1 - y_t) \sum_{j=1}^{n} \frac{\partial w_t^j x_t^j}{\partial \beta^i} - \mu (1 - z_t) y_t \sum_{j=1}^{n} \frac{\partial w_t^j x_t^j}{\partial \beta^i} \\
&\approx \beta_t^i + \mu z_t (1 - y_t) x_t^i \frac{\partial w_t^i}{\partial \beta^i} - \mu (1 - z_t) y_t x_t^i \frac{\partial w_t^i}{\partial \beta^i} \\
&= \beta_t^i + \mu \left[z_t (1 - y_t) - y_t (1 - z_t) \right] x_t^i h_t^i \\
&= \beta_t^i + \mu \delta_t x_t^i h_t^i .
\end{aligned}
$$

The derivative $\frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i}$ is approximated with the assumption that the effect on w_t^j due to changes in β^i for $i \neq j$ is small. This diagonalization allows us to avoid computing the Hessian on each timestep. Instead of this expensive computation, the accumulating trace, h_t^i is used here. Note that in the case of the Half-Moon World, where there is only one parameter, this derivation is exact.

In order to compute the β_t^i update, h_t^i must be calculated. A recursive function can be derived from the weight update equation given in Equation 4.3:

$$
\begin{aligned}
h^i_{t+1} &= \frac{\partial w^i_{t+1}}{\partial \beta^i} \\
&= \frac{\partial}{\partial \beta^i}(w^i_t + \alpha^i_{t+1}\delta_t x^i_t) \\
&= \frac{\partial w^i_t}{\partial \beta^i} + \frac{\partial \alpha^i_{t+1}\delta_t}{\partial \beta^i} x^i_t \\
&= h^i_t + \frac{\partial \alpha^i_{t+1}}{\partial \beta^i}\delta_t x^i_t + \alpha^i_{t+1} x^i_t \frac{\partial \delta_t}{\partial \beta^i} \\
&= h^i_t + \frac{\partial e^{\beta^i_{t+1}}}{\partial \beta^i}\delta_t x^i_t + \alpha^i_{t+1} x^i_t \frac{\partial z_t}{\partial \beta^i} - \alpha^i_{t+1} x^i_t \frac{\partial y_t}{\partial \beta^i} \\
&= h^i_t + \alpha^i_{t+1}\delta_t x^i_t - \alpha^i_{t+1} x^i_t y_t(1-y_t)\frac{\partial \mathbf{w}_t{}^T \mathbf{x}_t}{\partial \beta^i} \\
&\approx h^i_t + \alpha^i_{t+1}\delta_t x^i_t - \alpha_{t+1} x^i_t y_t(1-y_t) h^i_t x^i_t \\
&= h^i_t[1 - \alpha^i_{t+1}(x^i_t)^2 y_t(1-y_t)] + \alpha^i_{t+1}\delta_t x^i_t.
\end{aligned}
$$

The full algorithm for logistic IDBD is given in Figure 1.

Algorithm 1 Logistic IDBD

Initialize h^i_0 to 0, w^i_0 and β^i_0 as desired.
for each timestep t **do**
  $y \leftarrow \frac{1}{1+e^{\sum_i w^i x^i}}$
  $\delta \leftarrow z - y$
  for each weight i **do**
    $\beta^i \leftarrow \beta^i + \mu\delta x^i h^i$
    $\alpha^i \leftarrow e^{\beta^i}$
    $w^i \leftarrow w^i + \alpha^i \delta x^i$
    $h^i \leftarrow h^i[1 - \alpha^i (x^i)^2 y(1-y)] + \alpha^i \delta x^i$
  end for
end for

4.2 Step-size Adaptation in the Half-Moon World

The tracking agent in Section 3.1 uses a fixed α set by the programmer. Because the benefit of α depends on characteristics of the agent's experience, it might be useful to give the agent the ability to autonomously tune α. The logistic IDBD algorithm provides just this ability, allowing the agent to tune the step-size to adjust the learning rate appropriately.

The IDBD algorithm

In the logistic IDBD algorithm, presented in pseudo-code in Figure 1, the weight update occurs on every `look` timestep as for the non-adaptive learning rule used in Section 3.1, with one change. Instead of a fixed α, α_t is computed on each timestep using the exponential function and a new parameter β_t:

$$w_{t+1} = w_t + \alpha_t \delta_t \ldots \tag{4.5}$$

$$\alpha_t = e^{\beta_{t+1}}. \tag{4.6}$$

Before the weight update, the parameter β_t is itself updated with a similar learning rule:

$$\beta_{t+1} = \beta_t + \mu\delta_t h_t. \tag{4.7}$$

The meta-step-size μ affects how quickly β_t changes in response to experience. The error, δ_t, is the same as for the weight update—the difference between the target and predicted value, $\delta_t = z_t - y_t$. The trace h_t is a second new parameter that maintains a record of the effect changing the step-size has had on the error:

$$h_{t+1} = h_t[1 - \alpha_t y_t(1 - y_t)] + \alpha_t\delta_t \tag{4.8}$$

Task

The test environment was the Half-Moon World with a `look` probability of .5. The performance of an IDBD agent was contrasted with the performance of a fixed-α tracking agent to determine whether learning-to-learn can provide any appreciable benefit. The learning rule for a tracking Half-Moon World agent was adapted according the IDBD algorithm.

Experiment details and results

Three different sets of experiments were run. The first was a basic test of IDBD, to show whether or not it learns the best α value. The second was to determine the sensitivity of the IDBD algorithm to the choice of meta-step-size μ, showing the rate at which IDBD found the best α for several different choices of μ. The third was to show whether or not meta-learning can provide a benefit. In this experiment the mean loss of the IDBD algorithm was compared directly to the mean loss of an agent that tracks through fixed step-size as in Section 3.1.

Finding α

To evaluate the ability of the IDBD algorithm to find a good α parameter, single trials starting from different initial step-size settings were run past convergence, to 1,000,000 `look` timesteps. The starting step-size, α_0 values ranged from .1 to 10. All trials used a meta-step-size of 0.0001, so that the final α_T values were fairly consistent.

To determine whether the α_T found by IDBD was in the best range, the fixed-α experiments of Section 3.1 were repeated at finer increments, with α values incremented by .25 from 2 to 8. These results are displayed in Figure 4.1 and Figure 4.2.

The α values found by IDBD ranged between 4.7 and 4.9 in the last 10,000 steps, as shown in Figure 4.1. Larger initial values of α_0 converged to the correct range most quickly, while the smallest α_0 took around 500,000 steps to reach the final range. In the detailed evaluation of the fixed-α tracking agent, shown in Figure 4.2, the lowest loss occurred between 4.25 and 5.5. Significantly higher loss occurred for agents with $\alpha > 6$ and $\alpha < 3.5$.

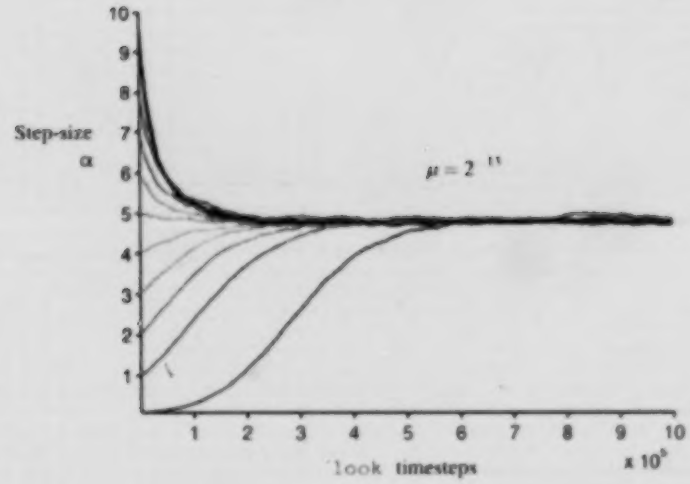


Figure 4.1: Illustration of the change in α_t over time with the logistic IDBD algorithm in the Half-Moon World. In the final 10,000 steps the α_t values stayed within 4.7 and 4.9 for all initial α_0 settings.

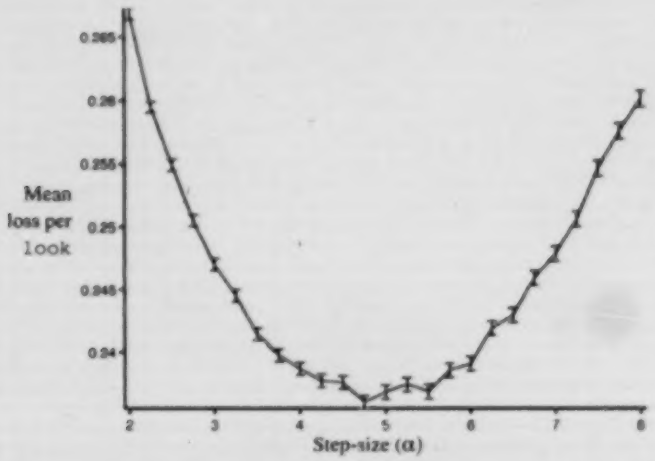


Figure 4.2: A more detailed look at mean loss for tracking agents with fixed step-sizes in the Half-Moon World (cf.Figure 3.3)

Meta-learning rate

To illustrate the effects of μ on the meta-learning rate, single trials of 1,000,000 look timesteps were run with six different μ settings ranging from 0.01563 to 0.00002 in powers of 4. The initial α_0 was set to 1. The step-size over time, α_t, was recorded and is shown in Figure 4.3.

Small values of μ resulted in slower convergence. Large values of μ resulted in high variance in α_t (Figure 4.3). After 1,000,000 steps, the agent with $\mu = 0.000002$ had not converged, with

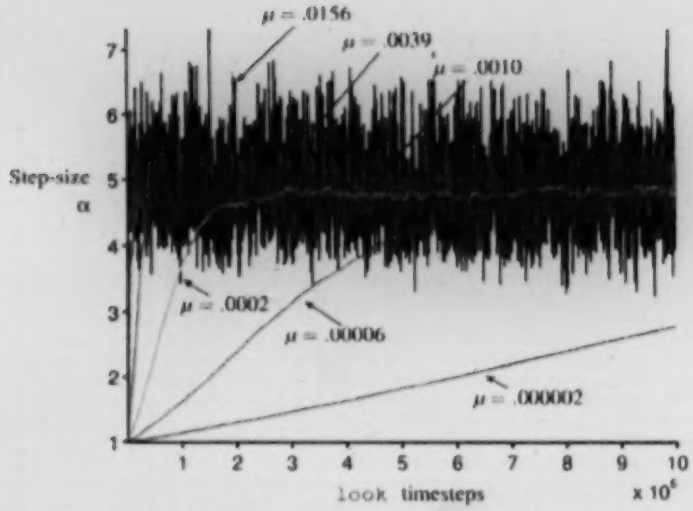


Figure 4.3: Illustration of the change in α_t over time for various settings of meta-step-size, μ.

$\alpha_t << 4$. For μ values greater than 0.0010, convergence of a sort happened within the first 100,000 steps, but the step-size values fluctuated by large amounts. In the case of $\mu = 0.0156$, the α_t values ranged between 3.2 and 7.4. In the case of $\mu = 0.0010$, the values ranged between 4.3 and 5.3. The μ setting that provided the most stable solution that still converged within 1,000,000, $\mu = 0.00006$, moved between 4.72 and 4.79 in the last 10,000 steps.

Loss comparison

To investigate the effect of the IDBD algorithm on training error, loss was averaged over 30 trials of 2,500 `look` timesteps for an IDBD agent with different α_0 initializations and a fixed-α tracking agent with corresponding α. The α values ranged from 0.0039 to 32 in powers of 2. The weight w_0 was initialized to -5. The mean loss incurred over the entire 2,500 steps was recorded and is illustrated in Figure 4.4.

In the comparison of the loss incurred by the IDBD and fixed-α agent, shown in Figure 4.4, the IDBD agent performed as well as or better than the fixed-α agent in all cases, though there was no statistically significant difference in their performances with $2 \leq \alpha \leq 8$. The improvement for $\alpha \geq 16$ was most significant, with the fixed-α agent having almost double the loss for $\alpha = 16$ and almost triple for $\alpha = 32$. The loss the IDBD agent incurred for $0.0156 \leq \alpha \leq 0.5000$ improved on the fixed-α agent by around .07 and this difference increased for α values smaller than 0.0313.

Discussion

The IDBD algorithm finds an α_t within the range that the fixed-α experiments showed is best for the Half-Moon World. An agent that learns to learn not only benefits from finding the best step-size

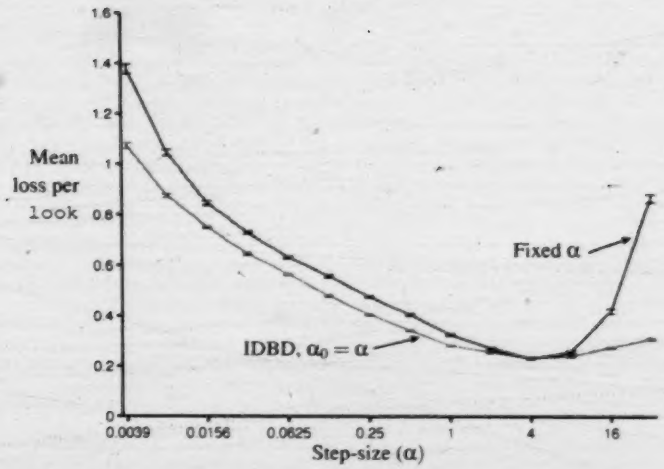


Figure 4.4: Comparison of the mean cross-entropy loss in the Half-Moon World from a fixed-α tracking agent and a logistic-IDBD agent. Loss is measured per timestep where the `look` action was taken. The darker line marks the loss of the fixed-α agent, using the α value indicated by the x axis, and the lighter the IDBD agent, with initial α set according to the x axis.

parameter, but also sees a reduction in loss.

The learning rate is affected by two things: the magnitude of μ, as shown in Figure 4.3, and the initial value α_0, as shown in Figure 4.1. The fact that IDBD recovers more quickly from α_0 values that are too large than from α_0 values that are too small may provide guidance when choosing initial parameters. The results in Figure 4.4 show that large α_0 do not greatly increase the mean loss. In addition, they show that setting α_0 to be too small can cause greater loss than setting α_0 too high, at least in the early stages of learning.

With a small meta-step-size, convergence is slow, but the final α_t value fluctuates within the range of values that perform well on the environment. With a large meta-step-size, α_t quickly changes, but the fluctuations are large and the extremes move beyond the best α_t values for the environment. A decaying meta-step-size might be particularly useful in this setting, to take advantage of the quick learning in early stages due to a large μ without suffering from the constant fluctuations in later stages.

4.3 Temporal Coherence for Task Transfer

Meta-learning is usually considered a second-order effect, which requires a long sequence of tasks before its benefits are clearly apparent. The results in Section 4.2 illustrate a large improvement in error even during the early stages of learning. The improvement was present over a range of initial α_0 settings, including those very close to the best hand-tuned values. One interpretation of these improvements is that there is, in fact, an infinite series of tasks. As the agent moves from the black

region to the white region, the prediction task changes continually. The environment itself only has twenty states, but because of the memory limitations of the agent, the agent is encountering a sequence that is impossible to represent and recall exactly. Some learning is transferred from one timestep to another when the region remains the same. At a higher level, the best meta-step-size transfers across the entire infinite series of tasks.

If the temporal coherence of the environment makes the benefits of meta-learning apparent, then in a temporally incoherent environment, the benefits provided by IDBD should not be as obvious. The following experiment compares the benefits of meta-learning in the temporally coherent Half-Moon World to meta-learning in a similar, but temporally incoherent, prediction task.

Task

In the temporally incoherent task, the agent must predict a sensation that is 1 or 0 with equal probability. There is only one state and the sensation received on one timestep is completely independent of the sensation received on the next. The best prediction is always 0.5.

The performance of an IDBD agent and a fixed-α tracking agent on the temporally incoherent task is compared to their performance on the temporally coherent task (the Half-Moon World with `look` probability of .5) in order to determine where meta-learning provides greater benefit.

Experiment details

A fixed-α agent and an IDBD agent were tested in both environments for α and α_0 values ranging from 0.004 to 8 in powers of 2. The Half-Moon World had a `look` probability of .5. In the Half-Moon World, the mean cumulative loss over the first 2,500 `look` timesteps was recorded and is reported with the standard error of the mean over 30 episodes in Figure 4.5. In the temporally-incoherent world, the mean cumulative loss over the first 2,5000 timesteps was also recorded and is similarly reported in Figure 4.6.

Results

For values of α from 0.0625 to .5000, there was no statistically significant improvement for the IDBD agent in the temporally incoherent world. For values of $0.0313 \geq \alpha \geq 1$, the IDBD agent had lower loss than the fixed-α agent. In the Half-Moon World, the IDBD agent had a statistically significant reduction in loss for all α values tested except for $\alpha = 4$ (This is the result seen in Figure 4.4 as well).

Discussion

Meta-learning with the IDBD algorithm reduces loss in both temporally coherent and temporally incoherent worlds. Adapting the step-size helps the agent recover from poor parameter settings, significantly improving the error when the initial α value is too high and slightly speeding up recovery

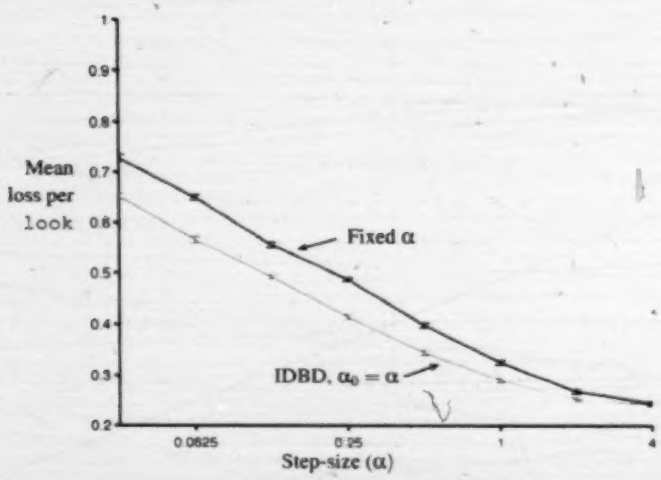


Figure 4.5: Comparison of the mean cross-entropy loss in the Half-Moon World for a fixed-α tracking agent and a logistic-IDBD agent. Loss is measured per timestep where the `look` action was taken. The darker line marks the loss of the fixed-α agent, using the α value indicated by the x axis, and the lighter the IDBD agent, with initial α set according to the x axis. The improvement due to IDBD is statistically significant everywhere but at the best α value (see Figure 4.4 for a wider range).

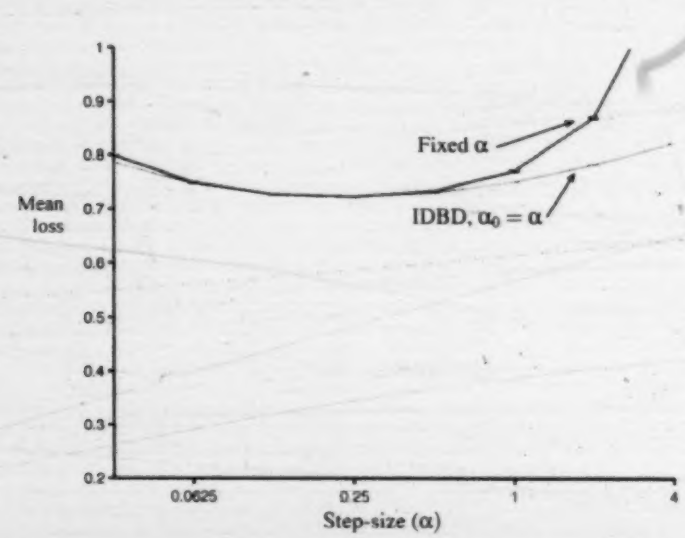


Figure 4.6: Comparison of the mean cross-entropy loss in a temporally incoherent prediction task for a fixed-α tracking agent and a logistic-IDBD agent. Loss is measured per timestep. The darker line marks the loss of the fixed-α agent, using the α value indicated by the x axis, and the lighter the IDBD agent, with initial α set according to the x axis. The improvement due to IDBD is only statistically significant for very large and very small α values.

from a poor initial w_0 when the α value is set low. Having a high α causes problems particularly in the temporally incoherent world, where convergence to a small range of values is important for reducing error. Having too small an α increases loss over the first 2,500 steps in both worlds, because the agent does not have time to recover from the poor initialization of $w_0 = -5$.

The difference between the IDBD and fixed-α algorithms arises not in the qualitative improvement, but in the extent of the effect. For extreme values in all worlds, IDBD allows the agent to recover from poor parameter choices. In the Half-Moon World, the improvement is apparent for all but the best α value. In the temporally incoherent world, the improvement is only significant for very wrong α values. Only in the high-loss case is there any significant advantage to using IDBD. Although the choice of step-size is important in the temporally incoherent problem, there is not the time to find it in the single, small, stationary task. Performance is dominated by the choice of the initial step-size (and initial weight), swamping the effect of meta-learning.

4.4 Conclusion

Being situated in experience means that learning how to learn can be an important part of an agent's knowledge representation. In AI and machine learning research, learning often refers only to learning the parameters of a function, with the form and meta-parameter chosen *a priori* or laboriously tuned. In fact, at some level there is no difference between learning that the parameter for some state variable should be high and learning that the parameter should adapt quickly. A continual-learning agent has the ability to meta-learn and tune its knowledge representation. The experiments presented in this chapter show that it can be beneficial to devote memory resources to meta-learning.

The Half-Moon World would not generally be thought of as a series of tasks appropriate for meta-learning, because testing the learning of meta-knowledge usually requires a long series of distinct tasks. A long series is usually required because the effects of meta-learning are dominated by the effects of learning parameters. Distinct tasks are considered necessary because retrieving a memorized solution does not count as transferring general knowledge to a new task. The experiments above show how a single environment might suffice for testing meta-learning and transfer. The temporally coherent regions of the environment provide an infinite series of distinct tasks as the agent moves between the regions of the world. The tasks are distinct because of the memory limitations imposed on the agent: it is not possible for the agent to memorize an exact solution. The agent must reconstruct the best solutions as it experiences a new region. Learning the best meta-parameters helps the agent to do so faster.

Chapter 5

Empirical Knowledge Representation

The following chapters recast some of the ideas already discussed and apply them to knowledge representation. Chapter 7 shows how the temporal coherence of predictions can lead to abstract knowledge representation. Chapter 5 presents a new framework for representing knowledge in terms of experience. Such a representation can use characteristics of experience, like temporal coherence, to represent abstract knowledge.

The term *empirical knowledge representation* is defined here as the class of knowledge representations that represent empirical knowledge. In an empirical knowledge representation, as in the empirical sciences, knowledge both describes and derives from verifiable experience rather than logical conclusions or theories. This tie to experience has two implications: first, all knowledge must be defined in terms of experience or in terms of other knowledge that is itself defined in terms of experience, and second, knowledge is predictive—it makes a verifiable statement about future experience. It is important to note that empirical knowledge representation does not demand that all knowledge be learned from scratch. Prior knowledge may be encoded in an empirical knowledge representation, but, whether learned from experience or built in, all knowledge in an empirical representation must be verifiable through experience.

The empirical approach to knowledge representation is unorthodox in the AI community, where the traditional approach to knowledge representation is not based in experience. Instead, knowledge is represented in a formal-logic framework. In this framework, all knowledge is symbolic, logical and completely separable from experience. A brief introduction to the traditional approach to knowledge representation is given in Section 5.1.

Empirical knowledge representation draws inspiration from the reinforcement learning framework. Reinforcement learning is a framework for solving decision-making problems by using experience. For an empirical knowledge representation, experience is the basis of knowledge. The similarities and differences between reinforcement learning and empirical knowledge representation are described in Section 5.2.

The key components of an empirical knowledge representation—experience, prediction and time—are described in Section 5.3. It is useful, in a representation built out of the raw data input to and output by the agent, to have a well-understood mechanism for abstraction. Two such frameworks are employed for the empirical knowledge representation presented in this chapter: options and predictive representations. Options are temporally extended actions, well-defined ways of behaving. An adaptation of the options framework is given in Section 5.4. Predictive representations provide the inspiration for the predictions that make up the core of an empirical knowledge representation. The formal definition of predictions used in this thesis is given in Section 5.5.

5.1 Non-Experiential Knowledge Representation

Knowledge representation is usually defined by the AI community in terms of formal logic and manipulation of symbolic entities (Russell and Norvig, 2003; Poole et al., 1998; Barr and Feigenbaum, 1981). In such symbolic logic systems, knowledge is considered to be logical statements about entities, where each statement has a true or false assignment. The statements are interpretable by humans as facts about the external world, but the system knows only symbols and relations. New knowledge must be manually added or uncovered by applying logical operators to the statements in the *knowledge base*—the database of symbols, relations and rules. This non-experiential approach has advantages and disadvantages, but it clearly emphasizes human-understandable meaning over machine-accessible data.

A prominent representative of the non-experiential approach to knowledge is Cyc, an encyclopedic knowledge base of 'commonsense knowledge' started by Lenat and collaborators, now run by Cycorp (Lenat et al., 1990). It is meant to contain the kind of knowledge that people draw on to interpret scenes, disambiguate sentence meaning and reason about behaviour. Cyc represents this kind of knowledge by encoding commonsense knowledge in a formal logical framework that has been carefully defined by humans. In order to be used by a decision-making agent, these symbols and relations must be translated into the framework used by the agent. Like other non-experiential knowledge representations, Cyc is built on the assumption that knowledge is separable from experience and that the question of relating agent-accessible data to knowledge is separable from the question of knowledge representation.

Encoding human-level knowledge in machine readable form requires humans translate their intuitions and knowledge into a special representation language (Matuszek et al., 2006). To add knowledge to Cyc, that knowledge must be fit into the allowed concepts. In Cyc, these concepts are hierarchical. Everything is a *Thing* and a *Thing* may be an *Intangible Thing* or an *Individual*. An *Individual* may have parts, but is not a *Collection* (a *Collection* is an *Intangible Thing*) (Cycorp, 2007). Encoding all possible relations and categorizations for every entity is difficult, but the difficulty is somewhat mitigated by the size of the knowledge base. A new fact might be that *JillBarber IsA Musician*. Because *Musician* is an entity already in Cyc, the symbol *JillBarber* may be auto-

matically categorized as a *Person*, *SocialBeing*, *Professional*, etc. This automatic categorization is one of the main goals of Cyc—to make the interpretation and acquisition of knowledge possible in a machine by drawing on a knowledge base that rivals that of humans. Such automatic categorization requires, however, that all the informal, fuzzy concepts human use be translated into strict relations and categorizations. It puts a burden on the designers of the representation to define the right categories and relations. New kinds of knowledge might require a great deal of restructuring of the database or require that knowledge to be force-fit into the existing framework.

Building and maintaining such a knowledge base requires extensive human input. More than 900 person-years have been spent entering commonsense knowledge into Cyc (Matuszek et al., 2006). The possibility of using machine learning for pulling information from the Internet or placing new facts in the hierarchy has been explored (Shah et al., 2006; Taylor et al., 2007), but the vast majority of data in Cyc has been entered by hand. New relations among existing facts can be discovered automatically through inference, but no new symbols or kinds of relations can be added automatically. The inability to create new symbols or kinds of relations is a direct consequence of Cyc's disembodied nature. Knowledge in the knowledge base just is: it is not tied to data outside the knowledge base. The lack of verifiability poses a difficulty for agent-driven acquisition of knowledge.

As with any large knowledge base, Cyc faces problems of brittleness. Inconsistent data is introduced to the knowledge base through typographical errors or mistakes in hierarchical classification. Besides these matters of data-entry error, there is the problem of unspoken or unrealized assumptions. One example of problematic assumptions is when the context of a fact is implicitly assumed. *Dracula IsA Vampire*, and *Vampires DoNot Exist*. Does Dracula exist? In the context of a novel or movie, yes. In other contexts, no (Lenat, 1995). To capture contextual errors, contextual information has to be added alongside the facts (Lenat, 1995; Taylor et al., 2007). Some errors can be caught by comparing against facts already in the database. For comparing new facts against existing facts, the size of the knowledge base is both a blessing and a curse: the more knowledge Cyc has, the more likely it is to know useful relations and accurately verify new knowledge. However, the more knowledge Cyc has, the more difficult it is to maintain consistency. Checking every new bit of knowledge against all existing knowledge and all knowledge that can be inferred is an extremely difficult and costly search problem. Cyc has long been too large for such exhaustive checking to be feasible, thus other methods for error detection must be used as well. The problems of brittleness derive at least in part from the dependence the knowledge representation has on human interpretation. In commonsense knowledge systems, the ultimate meaning of the knowledge is dependent on human interpretation. This dependence makes agent-driven evaluation of knowledge difficult.

5.2 Reinforcement Learning

Empirical knowledge representation draws inspiration from the reinforcement learning (RL) framework introduced in Section 2.2. The integral role of experience in reinforcement learning and the

emphasis on agent-directed learning in RL provides valuable insights for creating autonomous AI and empirical knowledge representations. In turn, an empirical knowledge representation framework might be particularly useful to reinforcement learning agents. The predictive nature of empirical knowledge means it is likely to be informative for an agent choosing actions to maximize future reward.

Reinforcement learning and empirical knowledge representation both have a mechanism for self-contained evaluation and share an emphasis on the primacy of experience. Experience is the fundamental unit of agent-environment interaction (illustrated in Figure 2.2) just as experience is one of the key components of an empirical knowledge representation. An RL problem always has a decision-making agent interacting with an environment. In both RL and empirical knowledge representation, an evaluation mechanism is directly accessible to the agent. The success of an RL agent is indicated by the reward it receives. The accuracy of an empirical knowledge representation is similarly evaluated against its experience.

Empirical knowledge representation departs from the the RL framework in a few significant ways. Reward is an essential signal in an RL problem. For knowledge representation, it is possible to develop a representation while treating reward as just another sensation or ignoring it entirely. Although the most successful approaches to RL use a value function to represent knowledge about anticipated reward, as discussed briefly in Section 2.2, an RL agent does not necessarily have a knowledge representation. Empirical knowledge representation does assume there is something to be gained in modeling and predicting experiential patterns.

5.3 Key Components of Empirical Knowledge Representation

An empirical knowledge representation is computed as a function of the past and makes predictions about the future. It is dynamic, responding to what the agent is experiencing now. It is verifiable, with all knowledge defined in terms of data accessible to the representation. The three essential components of an empirical knowledge representation are experience, time and prediction.

Experience is the sequence of sensations received and actions chosen by the agent, the data input and output. For a robot moving along the surface of Mars, the input is the sensations it receives from its cameras, spectrometers, thermometers and other sensors. The output signals to its various motors and joints. For a computer playing chess, the input is the position of the pieces on the board. The output is the movement of one of those pieces. This raw input and output is all the data that is accessible to the agent. What is meant by experience in an empirical knowledge representation is no more or less than the temporal sequence of data.

Experience is crucial for an empirical knowledge representation because it provides the means by which an agent can evaluate its knowledge. The meanings we humans ascribe to colours and temperatures and chess pieces are not part of the data another agent has access to, and so a knowledge representation that relies on those meanings can not be independent. An experience-based knowl-

edge representation is not reliant on external signals: it can develop independently of the designer. An empirical knowledge representation is thus equipped with a means for autonomous knowledge representation.

Time is an intrinsic characteristic of experience, but deserves particular mention because the consequences of a temporally situated knowledge representation are so often overlooked. The temporal situation of experience means that what is happening right now has particular relevance to the agent. It is *now* that the agent must chose an action. It is *now* that the agent is receiving sensations. As demonstrated in earlier chapters, the richness of immediate experience can be exploited to provide a knowledge representation with more detail than is possible in a knowledge representation divorced from time and averaged over all experience.

In its acknowledgment of the primacy of now and the inescapable passing of time, empirical knowledge representation stands in contrast to other approaches in learning and knowledge representation. Most modern learning theory is oriented towards a static knowledge representation that weights all experience equally. An empirical knowledge representation, in contrast, is continually learning. Continual learning can, as in the environments explored in Chapter 3, lead to a great improvement in performance and prediction accuracy.

The third component of an empirical knowledge representation is *prediction*. An empirical knowledge representation uses information computed from past experience to make predictions about future experience. These predictions can be understood as answers to questions about future experience: What is the probability I will see a wall if I move forward? Will making this move win the game? These questions are computed as a function of the current knowledge representation, answering a question about something still to come. They may directly estimate the probability of future events or estimate what other predictions will be: If I capture my opponents' piece, how will that change my expectation of the final score? An agent's empirical knowledge representation is stored in predictions.

Predictions are well-suited to knowledge representation because they are both informative and verifiable. Predictions are informative for decision making because they can explicitly encode the consequences of decisions; predictions can be conditioned on ways of behaving and predict relevant sensations (such as reward). They are verifiable against the future experience of the agent because they are defined in terms of experience. Section 5.5 formally presents how this verification and construction is done in an empirical knowledge representation.

5.4 Options

The options framework provides a formal mathematical framework for actions that extend over a period of time (Sutton et al., 1999). It was introduced as a way to add temporal abstraction to Markov Decision Processes. This section introduces the options framework and explains how it has been adapted for empirical knowledge representation.

Formally, an *option* is mathematically described by the functions π and β and the set $\mathcal{I}$. The option policy, π, like the regular policy described in Section 2.2, maps states variables $\mathbf{x}$ to a probability distribution over actions $a \in \mathcal{A}$. The termination condition, β, determines when the option is finished. It is a probability distribution over state variables, giving the probability in each state that the option will terminate. The initiation set, $\mathcal{I}$, describes when the option is available to be taken. It is the set of state-variable conditions in which the option may be started. Using options rather than primitive actions that extend over only a single timestep in a discrete dynamical system allows us to easily define temporally extended ways of behaving.

If an agent is provided with a useful set of options, it can learn a good solution faster than it could with only primitive actions. This speed-up can happen when the number of options between the current state and some goal is fewer than the number of primitive actions. Learning can also be sped up when the number of states where options need to be chosen is smaller than the number of states where individual actions need to be chosen because the options extend over more than one state. Options do not necessary extend over more than one state, however. The set of options may, and often does, include the set of primitive actions. Even when the set of options includes all primitive actions, learning can be sped up when the options provide ways of behaving that are more appropriate for maximizing reward (McGovern, 2002).

The options framework allows details of the agent's behaviour to be abstracted away when necessary. As long as the π, β and $\mathcal{I}$ can be defined in terms of the actions and state variables of the agent, then the option can be treated a single choice. Because the agent can have many different options that last for different and variable amounts of time, it is possible for the agent to both learn and plan at multiple time-scales (McGovern, 2002). This abstraction has the potential to allow the generalization of knowledge to new domains, for example from a training or 'sandbox' environment to a more complex one (Singh et al., 2005). However, traditional options can only transfer knowledge across domains where the state space is exactly the same. McGovern and Precup describe experiments where an option-conditioned value function is usefully transferred between domains, but those tests involved only moving the goal and changing action-success probabilities within a consistent state space (McGovern, 2002; Precup, 2000). Although this limited kind of transfer is certainly useful, transfer between domains with greater differences might be valuable. Konidaris and Barto have begun development on an *agent-space* options framework, where the options are defined over the parts of the state features that are consistent across tasks and has shown how these options can be transferred between similar domains (Konidaris and Barto, 2007). The options framework used in this thesis is defined over state variables, rather than a tabular state representation, and thus has the potential to transfer in a similar way.

The options framework used in this thesis is a simple extension of the options framework as used in the RL community. The functions and set that describe an option for empirical knowledge representation are defined over the internal state variables of the agent, rather than directly over the

sensation output by the environment. The application of options also differs slightly from the usual use of the options framework. The agent's policy is not necessarily defined over options in this thesis. Rather than directing the agent's behaviour, as in most RL applications of options, options are used in this thesis for abstraction. Whether or not the option is followed, predictions can be conditioned on options to provide a certain level of abstraction. This is explained in more detail in Section 7.3.

5.5 Predictions

A prediction is defined by two functions: the loss function and the answer function. The loss function implicitly defines the question a prediction is asking by defining how error is calculated with respect to future experience. The answer function computes an answer as a function of the state variables. These functions are illustrated in Figure 5.1. The loss function $\mathcal{L}(f_t)$ defines the error of the prediction with respect to the future f_t and is computed when the condition c_t is met. The target z_t is used in the learning rule to update the parameters. In the answer function the state-variable vector, $\mathbf{x}_t \in \mathcal{R}^n$, is combined with the learned parameters, $\mathbf{w}_t \in \mathcal{R}^n$, to make the prediction. The predicted outcome, $y_t \in \mathcal{R}$, is computed from some function of the state-variable vector, $\sigma(\mathbf{w}_t, \mathbf{x}_t)$.

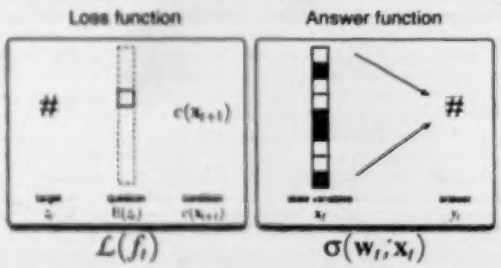


Figure 5.1: The anatomy of a prediction. The answer y_t is computed with function σ, combining the state variables $\mathbf{x}_t$ and parameters $\mathbf{w}_t$. The loss function provides the means for verifying the prediction against experience, defining the error between the answer and the target z_t when condition $c(\mathbf{x}_t)$ is met. The question induced by the loss function usually asks what the expected value of some future state variable is.

One example of a simple question is 'What am I likely to see on the next timestep?'. The loss function for this question reports the error between the predicted outcome and the actual value of the relevant sensation over time. In one common case, it computes the squared error, $(y_t - z_t)^2$. The cases presented so far use the cross-entropy loss, $-y_t \log(z_t) - (1 - y_t) \log(1 - z_t)$. A conditional version of the question is 'What am I likely to see on the next timestep if I turn right? The loss function $\mathcal{L}(f_t)$ defines the error of the prediction with respect to the future f_t and is computed when the condition c_t is met. The target z_t is used in the learning rule to update the parameters.'. The loss function would be similar to the unconditional case, except it would be undefined when the

condition was not met (*i.e.*, the action 'turn right' was not taken). A learning rule can be constructed based on the loss function and the predicted outcome function.

The Half-Moon World experiments presented in Section 3.1 can be recast using empirical knowledge representation. The state variable in the Half-Moon World is a constant term: $\mathbf{x}_t = [1]$. The loss function for the single prediction of the current region is the cross-entropy loss between the sensation on the next timestep and the prediction, y_t, with $c(a_t = \texttt{look}) = 1$. The answer function is the logistic sigmoid, $y_t = \frac{1}{1+e^{-\mathbf{w}_t^T \mathbf{x}_t}}$. The corresponding target, for the gradient-descent learning rule, is s_{t+1}.

This particular framework for prediction owes much to the recent work on predictive representations described in Chapter 6, but is formally presented and applied to knowledge representation for the first time here.

Chapter 6

Experiential and Predictive Knowledge Representation

Several branches of AI and cognitive science research have developed in reaction to the heavy emphasis on symbols in most approaches to knowledge representation. This chapter provides a brief introduction to AI research that emphasizes experience over abstract symbolic knowledge. In experiential AI, research is focused on systems that react to experience, sometimes rejecting representation entirely but always maintaining an emphasis on sensorimotor experience. A few key approaches to experiential AI are presented in Section 6.1.

Though the focus of reinforcement learning (RL) is on behaviour rather than knowledge, RL has inspired research on experiential knowledge representation through emphasis on the value function. Section 6.2 presents TD-Gammon, an RL agent that plays backgammon, as an example of how knowledge can be captured through value-function learning.

Recently, an empirical approach to state representation has arisen from the RL community. Predictive representations are very closely related to empirical knowledge representation. Predictive representations focus on learning and using predictions about future experience in order to model the state of a dynamical system. These predictive representations of state have also demonstrated the capability to encode certain kinds of knowledge. An introduction to work in predictive representations and how recent results relate to knowledge representation is given in Section 6.3 and 6.4.

6.1 Experiential Knowledge Representation

The historical emphasis on symbolic logic for AI resulted in a backlash against strong AI, the claim that a computational model of mind is possible. Philosophers claimed that systems that manipulated meaningless symbols could never be said to know anything and strong AI was doomed to failure (Searle, 1980). Partly in response to these criticisms, Harnad introduced "the symbol grounding problem": the question of how "the semantic interpretation of a formal symbol system [can] be made intrinsic to the system, rather than just parasitic on the meanings in our heads" (Harnad, 1990). This

question of making interpretation intrinsic to the AI agent is similar to the notion driving empirical knowledge representation: that knowledge must be representable in terms of data directly accessible to an intelligent agent. Indeed, most attempts to address the symbol grounding problem focus on tying representation to sensorimotor data (Taddeo and Floridi, 2005). However, the framework for empirical knowledge representation is not meant to directly address the symbol grounding problem. Discussions of the symbol grounding problem seem mainly united by the idea that there is a problem with symbolic representation, but with little agreement about exactly what the problem is or what the correct aim of representation is. Philosophical entanglements around meaning, knowing and consciousness make it difficult to address the problem of symbol grounding. Instead, this thesis is a look at what can be done with experience without attempting to answer the question of what it all truly means.

An alternative to the idea of knowledge as symbolic manipulation is the idea of *embodied cognition*, which arose from the cognitive science community. Embodied cognition describes learning and knowing as situated activities, not separable from a body's interaction with an environment (Anderson, 2003; Wilson, 2002). The emphasis embodied cognition places on environment interactions is similar to the emphasis in empirical knowledge representation on the primacy of experience. The aim and definition of knowledge in both empirical knowledge representation and embodied cognition is tied to the interactions between the intelligent agent and its environment. For both, knowledge itself depends on the actions taken by the agent as well as the sensations received and is not considered separable into an abstract, symbolic system. If symbolic manipulation can be thought of as divorcing knowledge from experience and narrowing cognition to consider only the symbolic processes within the mind of the agent, embodied cognition might be said to expand knowledge to encompass not just the agent with its inputs and outputs, but the entire world it lives in as well. Empirical knowledge representation falls between these extremes, insisting that experience is part of knowledge while allowing that the cognitive processes of the agent can be considered without direct reference to the external world. Embodied cognition emphasizes not just the inputs and outputs of a mind but also physicality and the reality of the environment the agent is situated in. In the most extreme view a disembodied computer can never be said to know anything, and strong AI is only realizable in a robot. In this thesis, experience is defined as the sequence of inputs to and outputs from an intelligent agent, rather than explicitly demanding a body interacting with a physical world. This understanding of experience allows us to discuss and experiment with virtual intelligent agents. The environment of the agent certainly effects cognition, but that effect is mediated by experience. Empirical knowledge representation assumes that defining knowledge in terms of experience is sufficient for capturing the situation and embodiment of cognition.

The most extreme kind of experiential AI is reactive systems. They dispense with representation altogether and "use the world as its own model" (Brooks, 1991). The concern in developing a reactive system is not the representation of knowledge, but the development of reactive behaviour mod-

ules and the subsumption architecture that determines how these modules interact. A less extreme version of reactive systems is behaviour-based systems (Matarić, 1997). Behaviour-based systems use the subsumption architecture of reactive systems, but individual behaviour modules may maintain internal representations. In both approaches, the system relies on simple behaviour modules that propose actions in direct response to experience. An empirical knowledge representation does not, obviously, dismiss representation altogether. Empirical knowledge representation and reactive or behaviour-based systems do however share the idea that knowledge apart from experience—knowledge-that is not empirical—is not absolutely necessary for an intelligent agent. Empirical knowledge representation differs from behaviour-based systems in that the empirical knowledge representation can be considered as a unified whole. Behaviour-based systems have decentralized control, with each behaviour module accessing only its own representation. The various predictions made by an empirical knowledge representation provide a kind of decomposition, but the state variables provide a centralized representation, available to any individual prediction.

6.2 Value Function Representation

One of the most successful approaches to reinforcement learning problems is to compute a value function as an intermediary between experience and a policy. Like the components of an empirical knowledge representation, the value function is computed from experience, tuned according to experience and makes a prediction about future experience—specifically, what the expected discounted sum of reward will be from a given state. The value function may be thought of as a special case of a prediction in an empirical knowledge representation. Flexible value function representations, such as neural networks, have been shown to be capable of representing interesting knowledge about a system (Orr and Müller, 1998). A specific example of this capability is described next.

TD-Gammon, the backgammon program developed by Gerald Tesauro, uses reinforcement learning with a multilayer neural network to represent the value function (Tesauro, 1995). A neural network is a data structure that can learn to model arbitrarily complex nonlinear functions. TD-Gammon's neural network takes information about the current board position as input and uses learned parameters to compute the value of that position.

The first version of TD-Gammon, TD-Gammon 0.0, used non-informative features that indicated the position of pieces on the board fairly directly (Sutton and Barto, 1998). It learned the parameters of the neural network by playing many thousands of games against itself. Initially, of course, TD-Gammon played very poorly, as moves were essentially random. However, after many thousands of games, TD-Gammon learned weights that could be seen as representing spatial information, giving high weight to favourably arranged board positions (Tesauro, 1992). The neural network used by TD-Gammon is a more complicated function than those discussed so far, but is a legitimate answer function, computable from experiential state variables (here, the features of the current board arrangement). TD-Gammon 0.0 thus is a simple empirical knowledge representation.

The second version of TD-Gammon, TD-Gammon 1.0, used handcrafted features. These features were still computed from the current board position but incorporated some expert knowledge (Tesauro, 1995). With this expert knowledge added, TD-Gammon became one of the strongest backgammon players, human or computer, in the world. The combination of expert knowledge for feature selection and experience-based knowledge of the correct parameters led to some surprising results. In contrast to the oft-cited claims that AI never leads to novelty, TD-Gammon's play overturned some world-champion wisdom about the best moves. Backgammon expert Kit Woolsey concluded that, unlike the chess computers that were strong on positions where tactical moves can be calculated out and weak on "vague positional games, where it is not obvious what is going on", TD-Gammon's "strength is in the vague positional battles where judgment, not calculation, is the key. There, it has a definite edge over humans" (Tesauro, 1995). In an empirical knowledge framework, state variables may be arbitrary functions of experience. As long as the expert-crafted features are computed from the board position, they can be included in an empirical knowledge representation. However, such features are not, themselves, empirical knowledge, as they are not verifiable against experience and make no statement about future experience.

6.3 Predictive State Representations

State representation is one of the fundamental issues in knowledge representation for reinforcement learning. Sometimes the state is directly observable, as in perfect-information games where the current position of pieces on the board is as informative as the entire history of the game. Other times, such as in the Half-Moon World of Chapter 3, the immediate sensations of the system do not provide a reasonable state representation. In these cases of partial observability, a state representation must be constructed (Shani, 2004). The state representation is a crucial part of the agent's knowledge.

Predictive representations use questions about future sensations as the basis for state. In the predictive representation framework, states are identified by the answers to a set of predictions about future experience. These predictions are estimated and updated online and the unique set of values provide a unique state label. Predictive State Representations (PSRs) are a particular method for representing state predictively (Littman et al., 2002; Singh et al., 2004). A PSR is a set of predictive tests, which can be informally thought of as questions: 'If I were to execute the specified sequence of actions, would I see the specified sequence of sensations?' When the specified sequence of actions results in the specified sequence of sensations, the test is said to *succeed*. When the specified sequence of actions does not result in the specified sequence of sensations, the test is said to *fail*. Answers that estimate the probability of success are computed on each timestep as a function of the current sensation, the current action and the answers from the previous timestep.

The focus in PSRs, and the difficulty, is to find the questions that are a sufficient statistic—the questions that, if the agent knows the answers, give the agent all the information it needs to distinguish between possible futures. This set of *core tests* provides the basis for the state representation:

the answers at any given time provide a unique identifier for the world state. If the test set forms a sufficient statistic, then they capture all distinguishing information possible (see Section 2.1). Theoretical work on PSRs has shown that PSRs are able to represent state for any dynamical system (Singh et al., 2004). The predictions in the empirical knowledge representation presented in this thesis are similar to the core tests of a PSR in that they are computed from experience and may make predictions about future experience. However, empirical knowledge representation research has a more general aim than representation state. In the empirical knowledge framework used in the next chapter, state variables that are not predictions are permitted and the set of state variables is not required to form a sufficient statistic.

Generalization and abstract concepts

The state of a dynamical system is an important piece of knowledge to represent, but for knowledge representation it is usually considered important to represent abstract or general concepts as well. In a recent paper, Tanner et al. used PSRs as a basis for supervised learning of human-labeled concepts (Tanner et al., 2007). The experience-oriented nature of the PSRs allowed these concepts to generalize to novel environments.

In the paper, gridworld-like maps were given two kinds of labels: automatically generated abstractions such as 'in a corner' or 'back to a wall' and human-labeled abstractions such as 'in a room' and 'in a corridor'. The agent had egocentric actions: the ability to move forward, turn right and turn left. A PSR was computed for the map and various numbers of core tests were used to learn a decision tree for each concept. The PSR-based decision tree achieved high classification accuracy on the test map.

This work illustrated that PSRs can be used as a basis for learning at least some of the abstractions that seem natural to humans, even when they are not obviously well-defined in experiential terms. It is similar to empirical knowledge representation in that the state variables used to compute the labels were predictive and experiential. It differs in that the knowledge represented by the system is not necessarily empirical. The evaluation could only be done against a human-labeled map. As in the case of TD-Gammon, the functions learned by this system are allowable state variables, though without verifiability they are not considered empirical knowledge.

Relational PSRs

Relational PSRs take a different approach to grounding abstract knowledge. In particular, Wingate et al. extend the traditional PSR framework with the introduction of wildcard tests, in order to capture relational knowledge, like that of formal logic frameworks (Wingate et al., 2007). Predictions, rather than only capturing state, are used to define various abstract concepts in grounded terms.

The test domain for Wingate's work on relational PSRs was a blocks world, a simple domain commonly used for planning with logical systems (Russell and Norvig, 2003). The blocks world

consists of a table with several cube-shaped blocks of different colours. Blocks can be moved onto the table or stacked on one another. The agent has an arm and an eye: a set of actions for picking up a specified block and putting it down and a set of actions for looking around. This domain can be (and usually is) described in completely symbolic terms with each block uniquely identified.

A relational PSR has two new types of tests and temporally extended actions, using the options framework described in Section 5.4. The usual test of a PSR is a sequence of actions and sensations. The two new test types of a relational PSR allow wildcards within the sequence of sensations. *Set tests* allow some sensations in the test sequence to be arbitrary. In order for a set test to succeed, the sensations following the specified actions must all match, except for the wildcard sensation, which matches against all actions. *Indexical tests* allow more than one of the sensations to be a variable. In order for an indexical test to succeed, every instance of the variable sensation must be the same (for example, whatever is observed on the third timestep must also be observed on the tenth).

The new kinds of tests allowed for a mapping between abstract relational concepts and the tests of the PSR. Knowledge such as 'block 12 is on block 14' and 'two blocks of the same colour are beside each other' has direct ties to actions and sensations. In the case of the first, when the predicted success of the test with actions 'find 12, look down' and sensations '12, 14' is 1, the agent knows that 'block 12 is on block 14'. Furthermore, using the set tests, the general concept of 'on' is representable with the action sequence 'find wildcard1, look down' and sensations 'wildcard1, wildcard2'. Direct mapping between symbolic relations and wildcard predictions is thus demonstrated as possible.

Though the components of relational PSRs are directly related to the components of an empirical knowledge representation, the aim is different. The work on relational PSRs aims to map symbolic logic onto an experiential framework. The investigation into experience of this thesis—particularly the explication of the idea of temporal coherence and the framework for empirical knowledge representation—is meant to illustrate how a focus on experience can lead to new insights about knowledge, rather than provide a mapping to the symbolic logic framework.

6.4 Temporal-Difference Networks

Temporal-difference networks (TD nets), like PSRs, represent state as a set of answers to questions about future experience (Sutton and Tanner, 2005; Tanner, 2005). A TD network is a network of nodes rooted in a sensation bit, updated through TD learning, as introduced in Section 2.2 and used in Section 3.3. The state variables in a TD net include the answers from each node from the previous timestep. They may also take include the last sensation, action or other features extracted from the history (Tanner and Sutton, 2005).

A TD net has two components: the question network and the answer network. A prediction is represented as a node in both the question and answer network. The question network defines the target of each prediction. Targets are usually the value of a sensation or prediction on the next

timestep, often conditioned on actions, as the prediction in the Half-Moon World in Section 3.1 was conditioned on the `look` action. The answer network defines the answer function, as described in Section 5.5, for each prediction. The answer function can be any arbitrary function of the state variables, but is usually linear or function (Tanner, 2005).

TD nets are a special case of an empirical knowledge representation, as the state variables are always computed as a function of experience and the answers are explicitly verifiable against future experience. Research on TD nets has mainly addressed the question of predicting sensations. Some applications of TD nets to the representation of other kinds of knowledge are presented next.

Generalization in TD Networks

Because a temporal-difference network is defined by the subjective experience of the agent, it has been proposed that TD networks are particularly well-suited to generalization (Rafols et al., 2005). This generalization hypothesis was tested by Rafols and colleagues in a simple gridworld environment, where states were grouped according to predictive similarity. This grouping "dramatically accelerated" learning (Rafols et al., 2005).

The idea that TD nets are useful for generalization starts with the observation that predictions that are conditioned on short sequences of actions may give identical answers in many different states, particularly in deterministic, fully observable environments. Similarly, for an action-conditioned TD net of a given length, there may be groups of states for which the answers computed by the TD net are identical. These sets of states are known as *identically predictive classes*. Identically predictive classes group states according to the predictions made from those states. Depending on the size of the TD net being used as the basis for grouping, the set of identically predictive classes can be considerably smaller than the set of full states. A tabular value function learned over the sets of identically predictive classes generalizes reward information between states within each class.

The generalization test domain in Rafols' work was a deterministic reinforcement learning task in an office-layout gridworld. The state was fully known, and the answers to the questions of the TD network were computed exactly from a model of the environment. For TD nets of several depths, the identically predictive classes were used as the basis for learning a tabular value function. Using small depths allowed for high generalization between states and therefore faster learning but less exact final solutions. As the depth of the TD networks increased, the generality of each predictive class decreased (and therefore the learning rate slowed), but the accuracy of the final answer increased. At a certain depth, the number of identically predictive classes equaled the number of states in the gridworld, with the slowest learning and best final answer (Rafols et al., 2005). The experiments demonstrated that generalization within an environment across identically predictive states can dramatically speed up learning.

Commonsense Knowledge

Further generalization and abstraction is possible in TD nets when action-conditioned links use temporally extended actions (such as options, described in Section 5.4). Rafols defined the structure and learning rule for an option-conditioned TD net (Rafols, 2006). In a simple, multi-sensation gridworld, the TD net captured some abstract and commonsense knowledge about the world.

In a multi-sensation world, a TD net is constructed for each sensation bit. Within each TD net, predictions are conditioned on the results of immediate actions and on the results of the results of options. The parameters of the TD nets are learned through interaction with the environment.

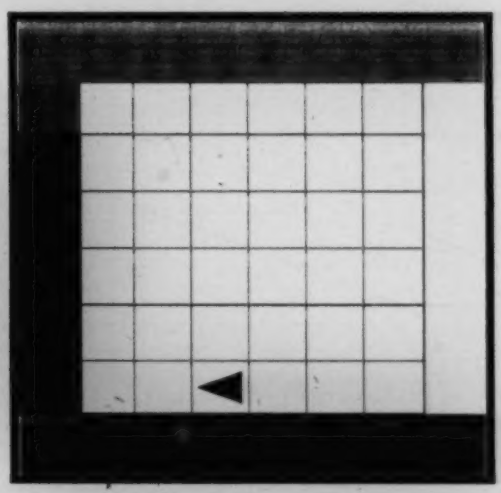


Figure 6.1: The multi-sensation compass world for testing option-conditioned TD networks. The agent, represented by the triangle, senses the colour of the square directly in front of it—this is usually white, but when the agent is directly adjacent to a wall, will be the colour of the wall. The primitive actions allow it to turn to the left or right or move forward.

The test domain for commonsense knowledge was a simple square gridworld with walls of different colours, illustrated in Figure 6.1 (Sutton et al., 2006). The agent's sensation on each timestep was a bit vector with one bit for every possible colour. When the agent was positioned directly adjacent to a blue wall, the blue bit was 1 and all the rest 0. When the agent was directly adjacent to an empty space, the white bit was 1 and the rest were 0. The agent had three primitive actions: turn left, turn right and move forward, and two options: `leap` and `wander`. The policy for the `leap` option moved the agent forward. It terminated when a colour other than white was observed and could be initiated in every state. Informally, the `leap` option allowed the agent to move forward until it hit a wall. The policy for the `wander` option was to choose actions randomly. It could be initiated from every state and terminated with probability 1 when a non-white colour was observed and with probability .5 otherwise. The `wander`-conditioned prediction thus represents knowledge

of whether a wall is near.

In a 8×8 version of the compass world, the agent trained for 250,000 steps. The predictions it learned allowed it to stay localized within the world, always knowing which wall it was facing through the `leap`-conditioned predictions. The predictions were maintained even when the agent moved around the middle of the world, where the immediate sensations were completely uninformative. Furthermore, it was possible for the agent to transfer learning to larger worlds, even worlds that were too large to learn from scratch (Rafols, 2006).

The application of PSRs and TD nets to more abstract kinds of knowledge than state representation and immediate predictions has been promising. These specific cases of empirical knowledge representation have provided inspiration for the general understanding of experience-driven knowledge representation this thesis has attempted to describe. Ideally, a general empirical knowledge representation framework will move AI research even further towards the ultimate goal of representing human-level knowledge. The next chapter uses the empirical knowledge framework described in Chapter 5 to explore how objects might be understood in empirical terms.

Chapter 7

Case Study in Empirical Knowledge Representation: Aspects of Objectness

Knowledge of the physical world is regularly assumed to include the notion of objects. In developmental cognition, for example, understanding objects as entities with independent external existence is considered a crucial step towards adult intelligence (Bower, 1977). This view of objects is a central part of what might be called the objective-reality view of knowledge, where experience (if considered at all) is a window to what is really out there. In contrast, the view taken in an empirical knowledge representation is that experience is what is really available, and knowledge can be directly concerned with experience rather than experience being a poor substitute for the real thing.

Taking account of experience is important, but an intelligent agent needs to move beyond the minutiae of experience towards abstract and theoretical thought. Humans certainly use the idea of things having independent existence and use that idea with some success. How can an empirical knowledge representation, with its commitment to representing knowledge in terms of the ever-changing and uninterpreted experience of the agent, represent abstract and theoretical concepts like the existence and permanence of objects?

The notion of objects, though key to objective-reality views of knowledge, is not itself perfectly understood. Philosophers, psychologists and, recently, computer scientists have attempted to define objects, but the notion has defied precise definition. Is an object that which has a physical structure independent of the perceiver (Spelke, 1994)? Or everything that can be assigned a symbol in formal logic (Markosian, 2000)? We know what we mean when we talk about objects, but what are they really? Representing such notions in a computer demands a precise definition, yet no definition has been found that is universally agreeable.

Rather than proposing another definition of objects and then mapping experience to the component parts, this chapter explores the experience of object knowledge by contrasting the objective-reality and empirical views of knowledge. The case study explores how ephemeral and uninterpreted

experience can shed some light on the our fuzzy notions of objectness and illustrates the first steps towards using empirical knowledge representation for abstract concepts.

The first three sections of this chapter build towards representing the knowledge of object permanence. The knowledge is split into three phases: existence, persistence and permanence. *Permanence* is used in the same sense as the well-known object permanence experiments, where a children's understanding of objects is measured by hiding the test object with an occluding object and measuring their reaction. For object permanence experiments with infants, looking time is used as a measure of surprise: if the occluding object is removed and the test object is not there, an infant who is thought to understand object permanence will look longer than when the occluding object is removed and the test object is still there. (Baillargeon et al., 1985). The original object permanence experiments of Jean Piaget had older children as subjects, and he supposed that where the child was looking for the object was an indication of how well he or she understood object permanence. Children who understood object permanence would look behind the occluding object to find the more desirable test object. Children who did not would look wherever the test object had previously been found, rather than where it was last seen (Piaget, 1954). *Persistence* is used here to describe permanence without an occluding object—rather than having the test object blocked by an occluding object, the test object disappears when the subject is looking away. *Existence* is used in this chapter to describe one of the most basic behaviours of objects—if I am starting at an object, it does not suddenly disappear. Section 7.1 illustrates how temporal coherence relates to this description of existence. Section 7.2 illustrates persistence and shows how the temporal coherence of predictions allow an agent to represent consistent experience even when the immediate sensations are not temporally coherent. Section 7.3 describes a full object permanence experiment, along with an explanation of how such knowledge might be represented in the empirical framework introduced in this thesis. Small computational examples are given for all three cases to illustrate how the temporal coherence of sensations and predictions relate to the notion of objectness.

Recognizing an object involves more than responding to a single, simple sensation. Section 7.4 and Section 7.5 show how the temporal coherence of more complicated patterns of experience can be identified in an empirical knowledge framework.

7.1 The Beginnings of Existence

Perhaps the most obvious feature of the existence of physical objects is that, when they are being observed closely, objects do not disappear. Consider the scenario illustrated in Figure 7.1. An infant is staring fixedly at a ball for some time. When the ball suddenly disappears, the infant registers surprise (Moore, 1975). One explanation of this surprise, from an objective-reality view, is that the infant recognizes the ball as a separate physical entity. As the ball is a physical object, it cannot just disappear. That is not what things with independent physical existence do.

An empirical knowledge representation frames this knowledge differently. The sensation being

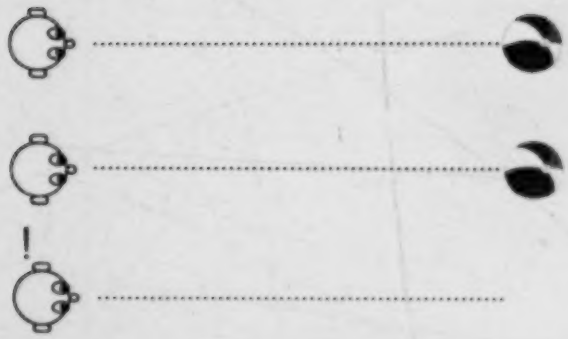


Figure 7.1: An infant registers surprise if the object disappears suddenly.

received by an infant staring fixedly at a ball has high temporal coherence. When the ball disappears, it violates the prediction that this sensory information should be consistent from timestep to timestep. That prediction of high temporal coherence encodes knowledge of how ball-sensory-patterns should behave.

A simple example of how temporal coherence is represented in this empirical knowledge framework can be illustrated by a extremely simple environment with no actions and a single bit sensation that changes probabilistically over time. This is a simplest representation of the situation described above, where the infant is staring at the same thing and taking no actions. Table 7.1 illustrates two trajectories: one with a uniformly random sensation, which is independent on each timestep, and one with a temporally coherent sensation, which is constrained to change only rarely.

Uniformly Random Bit		Temporally Coherent Bit	
Sensation	Prediction	Sensation	Prediction
0	0.50	1	0.88
0	0.50	1	0.92
0	0.50	1	0.94
0	0.50	1	0.95
1	0.50	1	0.96
0	0.50	1	0.97
1	0.50	1	0.97
1	0.50	0	0.40
1	0.50	0	0.12
1	0.50	0	0.08
1	0.50	0	0.06
0	0.50	0	0.05
1	0.50	0	0.04
0	0.50	0	0.03
1	0.50	0	0.03
0	0.50	0	0.03
0	0.50	0	0.02
1	0.50	0	0.02
1	0.50	0	0.02
1	0.50	0	0.02

Table 7.1: Sample trajectory for a uniformly random bit sensation and a temporally coherent bit sensation.

Knowledge of the temporal coherence of the bit is represented in the parameters of a predictive state variable. The predictive state variable uses a logistic function to minimize the cross-entropy loss between the sensation on the answer from the previous timestep, as in Section 3.2. The answer is the expected value of the bit. The best answer depends on the long-term probability of the bit being on and, if the bit has temporal coherence, on recent experience. If the agent knows the bit has no temporal coherence, then the answer it computes will be consistent from timestep to timestep, with no weight given to recent experience. If the agent knows the bit has high temporal coherence, then it will put heavy weight on the most recent sensation when it is computing the expected value.

This knowledge is illustrated in the prediction columns in Table 7.1. For the uniformly random bit, the predictive state variable places no weight on the sensation of the previous timestep, predicting the same value regardless of the previous sensation. In the temporally coherent case, the expected value of the bit fluctuates according to recent experience. The parameters of the predictive state variable thus represent the tendency of the sensation bit to remain the same.

The prediction of the temporally coherent sensation gives us a starting point to represent a simple understanding of existence, the knowledge that the sensation should be consistent from timestep to timestep. The higher the temporal coherence of the bit is known to be, the more surprising a sudden change will be. In contrast, in the uniformly random case, where the prediction has maximum entropy, frequent changes are expected. Empirical knowledge representation can capture both temporally coherent and temporally incoherent sensations.

7.2 Persistence

A further step towards object permanence is the recognition that objects persist even when the sensation does not. The immediate sensations of an agent looking at and away from an object do not have temporal coherence, but there is a consistency in the sensations that it can be useful to represent. Consider the scenario in Figure 7.2. Very young infants are surprised if the ball disappears before their eyes, but are not surprised if the ball disappears while they are looking elsewhere. Older infants do register surprise when they look back and the ball is no longer there (Ginsburg and Opper, 1969). The usual description of this development is that older infants know that the ball is still there, even though they are not perceiving it directly (Bower, 1977). They know that the ball continues to exist independently of current sensations.

In an empirical knowledge representation, this knowledge is again represented in the predictions. This time, the knowledge that the ball should still be there after looking away is captured in a prediction conditioned on looking. Even when the immediate sensations change, accurate predictions of 'what would happen if' are temporally coherent.

A simple encoding of this knowledge is illustrated in Table 7.2. This environment allows multiple actions. The first action, a_1, behaves as in Section 7.1, returning a temporally coherent sensation bit. All other actions, $a_2 ... a_n$, result in a sensation of 0. These actions are like looking in different

Figure 7.2: An infant registers surprise if the object disappears while the infant is looking away.

directions, most of which are boring. The conditional sensation generated by the environment during a sample trajectory is given in the first column. This column shows the sensations that would result if the agent took the specified action. The second column shows the sensation actually received by the agent, given the action that was chosen.

Conditional Sensation				Actual Sensation				Conditional Prediction
a_1	a_2	a_3	a_4	a_1	a_2	a_3	a_4	$Pr(s=1\|a=a_1)$
1	0	0	0	1				0.88
1	0	0	0		0			0.88
1	0	0	0				0	0.87
1	0	0	0	1				0.92
1	0	0	0	1				0.94
1	0	0	0	1				0.95
1	0	0	0			0		0.94
0	0	0	0		0			0.94
0	0	0	0		0			0.94
0	0	0	0		0			0.93
0	0	0	0				0	0.93
0	0	0	0		0			0.93
0	0	0	0	0				0.23
0	0	0	0		0			0.23
0	0	0	0			0		0.24
0	0	0	0		0			0.24
0	0	0	0			0		0.25
0	0	0	0	0				0.11
0	0	0	0	0				0.08
0	0	0	0	0				0.06

Table 7.2: A comparison of the conditional sensation, the sensation the agent would receive if it were to take action 1 on each time step, to the actual sensation and prediction. The actual sensation is what the agent receives given it is randomly selecting actions, and the conditional prediction is the value of the predictive state variable that predicts the sensation after the search option is executed.

The agent maintains a predictive state variable that minimizes the log loss between the computed answer and the sensation received upon taking action a_1, much like the prediction made by the Half-Moon World agent from Section 3.1. The last column in Table 7.2 indicates this prediction of 'What would the sensation be if I were to take action a_1?' With both tracking and transience,

as in Section 3.2, the prediction updates towards the most recent sensation whenever a_1 is taken and decays towards the long-term average when a_1 is not taken, as then the prediction is not being frequently verified.

The size of the predictive state variable indicates knowledge of something beyond the current sensation, the expected value of the bit if a_1 is taken. The persistence of that prediction can be represented regardless of what action the agent is taking. The example given uses the strict memory limitations of Section 3. There is no reason, of course, that the agent should not be able to predict the sensation bit exactly. In this example, the bit is temporally coherent but stochastic. In an environment where the sensation being returned was deterministic, the action-conditioned prediction could exactly predict the sensation bit.

7.3 Permanence

Child-development pioneer Jean Piaget considered that proper development of the object concept required actively looking for the missing object: passive observation, waiting for the object to reappear in the same place, develops into active looking for the object, even to the point of moving obstructions (Piaget, 1954; Ginsburg and Opper, 1969). A simple occlusion experiment is illustrated in Figure 7.3. At a young age, children do not register surprise if the ball disappears while hidden or even if it changes into another thing entirely (Bower, 1977). Older children are surprised when the obstruction is removed and the ball is missing or transformed into a train or a box. The objective-reality explanation is similar to before. Older children know that the ball is still really there behind the screen. They know that objects do not change form or disappear even when they are not being observed (Baillargeon, 1999). Younger children do not know that objects do not change form or disappear and are not surprised when the boundaries of objects shift or transform while they are hidden.

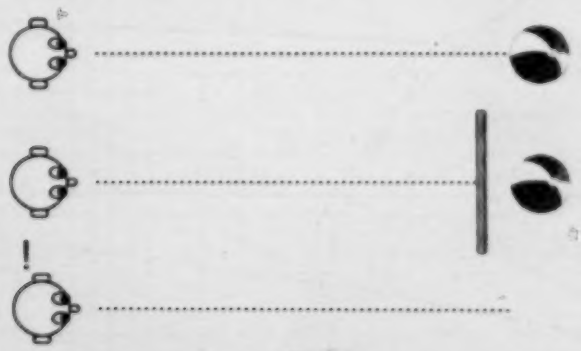


Figure 7.3: An older infant registers surprise if an object disappears while it is hidden.

The empirical account describes this knowledge differently. Knowing that there should be something behind the screen and that the something should still be a ball is knowing that the predicted sensations should be the same before and after the screen is in view. Rather than simply being conditioned on a single action, the prediction now must have a more complex condition to represent the movement of the occlusion or the action of looking behind the screen.

The options framework gives us the complexity needed to handle this case. Whereas before the agent maintained a prediction conditioned on a single actions, now the agent maintains a prediction conditioned on an extended way of behaving. For the simplest illustration of extended predictions, the environment described in Section 7.2 is extended by removing the restriction that the 1 bit is only returned following a_1. On any given timestep, the 1 may be sensed following any one of the actions or none. The 1 still has temporal coherence: if any particular action results in the sensation of 1, taking that action again will, with high probability, result in the sensation of 1. Taking a different action will, with high probability, result in the sensation of 0. A sample trajectory is illustrated in Table 7.3. The sensation the environment would return following each action is given in the first column, while the actual sensation received by the agent is given in the second column.

Conditional Sensation				Actual Sensation				Search-Conditional Prediction
a_1	a_2	a_3	a_4	a_1	a_2	a_3	a_4	$Pr(s = 1 \mid option = \texttt{search})$
0	1	0	0	0				0.50
0	1	0	0		1			0.88
0	1	0	0				0	0.88
0	1	0	0	0				0.87
0	1	0	0	0				0.87
0	1	0	0	0				0.86
0	1	0	0			0		0.86
0	1	0	0		1			0.91
0	1	0	0		1			0.93
0	1	0	0		1			0.95
0	1	0	0				0	0.94
0	1	0	0		1			0.95
0	1	0	0	0				0.95
0	1	0	0		1			0.96
0	1	0	0			0		0.95
0	1	0	0		1			0.96
0	1	0	0			0		0.96
0	1	0	0	0				0.95
0	1	0	0	0				0.95
0	1	0	0	0				0.95

Table 7.3: A comparison of the conditional sensation, the sensation the agent would receive if it were to take action 1 on each time step, to the actual sensation and prediction. The actual sensation is what the agent receives given it is randomly selecting actions, and the conditional prediction is the value of the predictive state variable that predicts the sensation after the search option is executed.

Suppose there is an option-conditioned prediction where the option is a random search procedure: for example, the option can be initiated at any time, the policy randomly selects among the available actions, and it terminates after all actions have been taken a specified number of times or the 1 has been sensed. If the option terminates on a 1 sensation, the 1 is present. If it terminates on 0, the 1 is very likely absent. Thus, a prediction of the terminal sensation of that option gives us an

indicator of the presence or absence of the 1.

Predictions conditioned on search procedures gives us something like object permanence. The agent 'knows the ball is there' when it has a prediction that a search procedure will end in the ball sensation. The search procedure in the above example is very simple, but it may be as complicated as necessary. Knowing the ball is there means knowing there is something I could do (or get someone else to do) to make the ball appear. Through the options framework, this kind of knowledge can be encoded as predictions of future experience, conditioned on arbitrary behaviour. The agent does not necessarily have to execute the behaviour to represent the prediction. Through the temporal coherence of option-conditioned predictions, knowledge that moves beyond the immediate sensations into abstract concepts can be clearly represented.

7.4 Configural Patterns

It is relatively straightforward to extend the single bit of the previous examples to more complicated inputs. Sensory input in robots and people is usually more than a single binary sensation and not all temporally coherent patterns can be identified by a single bit. In many cases, the single units of an agent's sensations are not temporally coherent, but configural patterns of the sensations are.

A simple example is illustrated in Table 7.4. An objective view of the environment is shown in the first column, with the part visible to the agent surrounded by a dotted line. The consistent pyramid pattern is completely visible to the agent on the first two steps and partially visible on the last. The actual sensation vector received by the agent is shown in the second column. There is little temporal coherence in the individual elements of this vector, but there is temporal coherence in the overall pattern. The agent has a pattern-recognition function that returns 1 when the pyramid pattern is present in the sensation vector and 0 otherwise. The output of this function is given in the third column. This output is more temporally coherent than the raw sensations, but still does not represent the consistency seen in the objective view. As in Section 7.3, the temporal coherence of the pattern can be represented with an option-conditioned prediction, shown in the fourth column. Knowledge of the pyramid's presence is encoded in a search-conditioned prediction, where the target of the prediction is the output of the pattern-recognition function rather than an immediate sensation.

The temporal coherence of configural patterns can thus be represented by defining a recognition function that returns 1 when the sensory inputs match a specified configuration and 0 otherwise. By considering the temporal coherence of the output of this function, persistence and permanence can be represented in the same way as in the single-bit case. The pattern-recognition function can be arbitrarily complex, indicating any patterns in the sensations received by the agent. The state variables used by the RLGO agent introduced in Section 2.4 are examples of one kind of pattern recognition.

Objective View	Actual Sensation	Pattern-Recognition Output	Search-conditioned Prediction
	0000000000100111	1	1
	0000000001001110	1	1
	0000000010001100	0	1

Table 7.4: A comparison of an objective view of the environment, the sensation vector received by the agent and recognition function for a pattern of sensation bits.

7.5 Predictive Patterns

Predictive state variables give an empirical knowledge representation powerful tools for representing complex patterns. What if the distinguishing features of an object can not be perceived all at once? This is the case illustrated below.

The environment in Figure 7.4 has two distinguishable shapes, E and F. As before, the agent's sensation on each timestep is a bit vector that maps onto a 4×4 square. This area is not large enough to always distinguish between the two patterns. An example ambiguous sensation is illustrated in Figure 7.5. The agent may have a recognition function for the two shapes, but there is not enough information in the immediate sensation to recognize E and F directly. Pattern-recognition over all the state variables, not just the immediate sensations, can address this case. With the addition of predictions to the pattern-recognition function, the agent can represent both E and F in this world.

The disambiguating prediction is illustrated in Table 7.5. The first column shows the objective view, the second the ambiguous sensation the agent is receiving. The agent has a special search option that can be initiated by the ambiguous sensation and terminates when the last four bits are 0, with a policy that causes it to look for disambiguating sensations. One way to represent the differences between the letters is to have two predictions conditioned on this search option, one predicting the probability of experiencing the pattern that belongs to the E, shown in the top row of the third column and one predicting the probability of experiencing the pattern that belongs to

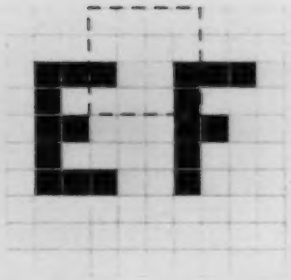


Figure 7.4: The objective view of a letter gridworld.

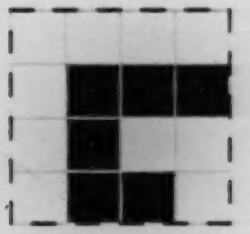


Figure 7.5: A recognition function that only looks at pattern of immediate sensations cannot necessarily disambiguate the F from the E.

the F, shown in the bottom. The recognition function for the letter E is active when the ambiguous sensation and the E-prediction are active at the same time and the recognition function for the letter F is active when the ambiguous sensation and F-prediction are active, as shown in the last two columns.

Extending the pattern-recognition function so that it includes predictive state variables extends the representational power of the agent's knowledge. In more complex environments, more expressive patterns are possible. For example, 'What will I sense if I eat this?' captures knowledge of tasty things, nourishing things, poisonous things and inedible things. Functional predictions provide a way of categorizing experience into useful groups. Predictive state variables give us a way of extending temporal coherence beyond immediate sensations, which in turn gives us the mechanisms necessary for the representation of knowledge about complex patterns of experience.

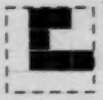
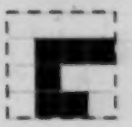
Objective View	Actual Sensation	Search-conditioned Prediction	E Pattern-recognition output	F Pattern-recognition output
E F			1	0
E F			0	1

Table 7.5: The predictive state variable disambiguates between the E and F when the immediate sensation is ambiguous.

7.6 Conclusion

The ideas presented above are meant to explore how empirical knowledge allows for the representation of abstract concepts, rather than provide a full model of the object-concept. We have only begun to test the limits of empirical knowledge.

The clear definition of knowledge has great potential for the development of autonomous artificial intelligence and our understanding of human cognitive processes. Predictions and temporally extended actions are well-defined in terms of the experience, and this chapter illustrates how they might enable the representation of abstract knowledge. It is possible the representation of abstract knowledge might be taken further. Empirical knowledge representation may provide an explanation for the close interaction between knowledge and sensation that others have observed in humans (Spelke, 1991; Spelke and de Walle, 1993).

AI research can fruitfully draw inspiration from the cognitive development of humans. This chapter models the progression of empirical knowledge towards the abstract along the lines of human development and shown how the combination of options, predictions and pattern-recognition might lead towards human-level cognition. Spelke suggests that infants' understanding of the physical world is centered around "divid[ing] perceptual arrays into units that move as connected wholes, that move separately from one another, that tend to maintain their size and shape over motion and that tend to act upon each other only on contact" (Spelke, 1990). It is just this kind of decomposition that an empirical knowledge representation is suited for.

Knowledge of the external world, such as understanding the nature of physical objects, has been thought difficult for subjective representations to capture. The power provided by predictive state variables and temporal abstraction through the options framework makes the first steps towards such abstract knowledge.

Chapter 8

Conclusion

This thesis marks the beginning of an investigation into the commitment to experience as the fundamental data. The results presented here are that dynamic memory can be more accurate than the best stationary solutions, that learning to learn can be essential to successful knowledge representation and that a framework for knowledge representation that is grounded in experience can capture at least some abstract knowledge. Chapter 2 introduced the idea of temporal coherence as a critical characteristic of experience. Experiments in a simple prediction task and the difficult problem of Computer Go showed how temporal coherence enables continual learning agents to adapt to their immediate environment and improve on their stationary counterparts. Chapter 5 provided a formal explanation of the framework for empirical knowledge representation: a representation that is grounded in experience and has the potential to answer the question of how an artificial agent should represent knowledge. A case study of how this framework might represent the object concept showed how empirical knowledge might allow for the representation of abstract knowledge.

8.1 Temporal Coherence

Current machine learning techniques are dominated by the idea of convergence to a single solution. Learning and the use of learned knowledge are clearly separated in the training and testing phases. In some sense, the product of learning has become more important than the process itself. Valuing the product above the process is a reasonable formulation for supervised learning tasks: the point of learning a classification through labeled training data is so that it can be applied to unlabeled data. In this case, there is a natural separation between the training and testing phase.

The distinction between the training and testing phases is not necessary to the reinforcement learning framework: the reward signal, which guides the learning in the training phase, continues to be available during any operation of the system. Even so, the majority of work in reinforcement learning applications retains the training/testing split of supervised learning. The split is done explicitly by converging to a single best answer over the course of training or by freezing the parameters after a certain number of timesteps. It is also implicitly maintained through a training phase that is

much shorter than the testing phrase or a learning rate that decays so experience gained during the testing phase has little effect on the agent's knowledge.

The results in Chapter 3 and Chapter 4 present the benefits of a different model, in which *now* has special emphasis. Past experience is used to make accurate predictions about future experience, much like training samples are used to make predictions on test samples. At any given point, the learned product represents the knowledge of the system. But rather than a rigid division between the past training phase and future test phase, there is the fluid tie to current experience. The agent does not learn for some number of steps and then declare its knowledge complete; it uses long- and short-term memory to continually learn, adapt and improve.

Why is it so important that *now* be treated as more than the dividing line between learning and knowing? *Now* is when a decision needs to be made, an action chosen. *Now* defines the state to which knowledge must be applied. Thus the knowledge representation needs to be tuned to the experience of *now*. If the distribution of every future from every state could be perfectly known and perfectly represented and the state itself could directly observed or computed, then it might not matter what is happening *now*. With perfect knowledge the agent could perfectly remember everything it needs to know about the state it is in now, as it perfectly remembers everything about every state. Such a perfect memory is rare and may not even be desirable, as imperfect memory allows for the transfer of knowledge to new situations. Aliasing between states can be useful for generalization, even when the state space is immediately observable to the agent and small enough to fit in a modern computer.

8.2 Empirical Knowledge Representation

What does it mean to know things? This question is fundamental to philosophy, psychology and artificial intelligence. Centuries of thought have not produced a consensus on the answer, and a definitive answer may not be possible. Is knowledge justified true belief? Then what is truth? And what sort of justification is sufficient? Does knowing about the world mean knowing how it really works down to the subatomic level—and beyond, if necessary—or does making accurate predictions suffice? Is real knowledge innate or learned or taught? And what is reality? These questions have prompted a wide variety of answers. But in the end, to develop an autonomous AI, we may have to set aside questions about what everything really means and look instead at what can be successfully used by AI agents.

The practical issue for artificial intelligence is how a machine can best represent knowledge. The choice of representation determines what is possible and impossible; easy and hard. It determines how well existing knowledge can be evaluated against new data and how robust the representation is to mistakes. It determines what the machine can learn and how new knowledge can be discovered and incorporated into the representation. It determines whether or not the representation can be adapted to match experience. A representation that is meaningful to the machine allows knowledge

to be automatically verified, tuned and even acquired.

The approach to meaningful knowledge representation advocated in this thesis is to use the immediately accessible data, the input and output, to build a representation that is informative for the decisions being made. The empirical knowledge representation presented in Chapter 5 is focused on the subjective and concrete experience of an agent rather than logical and abstract statements about an external world. Chapter 7 showed how it might be possible to represent some abstract knowledge in this empirical framework.

Empirical knowledge is based on and verifiable through data immediately accessible to the agent—its experience. Empirical knowledge is stored in and constructed from experience and predictions of future experience. It is continually updated in response to current sensations. It does not require an external interpreter and is not reliant on an external mind assigning meaning. We propose that knowledge is rightly understood in experiential terms, and that pursuing empirical knowledge representation holds the key to creating autonomous artificial intelligence.

Bibliography

Anderson, M. L. (2003). Embodied cognition: A field guide. *Artificial Intelligence*, 149(1):91–130.

Baillargeon, R. (1999). The object concept revisted: New directions in the investigation of infants' physical knowledge. In Margolis, E. and Laurence, S., editors, *Concepts: Core Readings*, pages 571–612. MIT Press, Cambridge, MA.

Baillargeon, R., Spelke, E. S., and Wasserman, S. (1985). Object permanence in five-month-old infants. *Cognition*, 20(3):191–208.

Barr, A. and Feigenbaum, E. A., editors (1981). *The Handbook of Artificial Intelligence*, volume I. HeurisTech Press, Stanford, CA.

Bower, T. G. R. (1977). *A Primer of Infant Development*. W. H. Freeman and Company, San Francisco, CA.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Inteligence*, 47(1–3):139–159.

Caruana, R. (2005). Inductive transfer retrospective and review. Slide Presentation at the NIPS 2005 Workshop on Inductive Transfer 10 Years Later.

Cycorp, I. (2007). Cycorp, inc. `http://www.cyc.com/cyc/technology/whatiscyc_dir/whatdoescycknow`.

Dailey, D. (2007). Cgos - a go server just for computers. `http://cgos.boardspace.net/`.

Gelly, S. and Silver, D. (2007). Combining online and offline learning in uct. In Ghahramani, Z., editor, *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML 2007)*, pages 273–280, New York, NY. ACM Press.

Ginsburg, H. and Opper, S. (1969). *Piaget's Theory of Intellectual Development: an Introduction*. Prentice Hall, Englewood Cliffs, NJ.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1–3):335–346.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY.

Higgins, J. J. and Keller-McNulty, S. (1995). *Concepts in Probability and Stochastic Modeling*. Wadsworth Publishing Company, Belmont, CA.

Konidaris, G. and Barto, A. (2007). Building portable options: Skill transfer in reinforcement learning. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 895–900, Rochester Hills, MI. IJCAI.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4: Proceedings of the 1991 Conference (NIPS 1991)*, pages 950–957, San Francisco, CA. Morgan Kaufmann Publishers, Inc.

Lenat, D. B. (1995). Cyc: a large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):32–38.

Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M. (1990). Cyc: toward programs with common sense. *Communications of the ACM*, 33(8):30–49.

Littman, M., Sutton, R. S., and Singh, S. (2002). Predictive representations of state. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference (NIPS 2001)*, pages 1555–1561, Cambridge, MA. MIT Press.

Macintyre, B. (2007). Why computers can't surpass go and collect $1 million: There's one game where artificial intelligence can't beat humans. `http://www.timesonline.co.uk/tol/comment/columnists/ben_macintyre/article2002699.ece`.

Markosian, N. (2000). What are physical objects? *Philosophy and Phenomenological Research*, 61(2):375–395.

Matarić, M. J. (1997). Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2–3):323–336.

Matuszek, C., Cabral, J., Witbrock, M., and DeOliveira, J. (2006). An introduction to the syntax and content of cyc. In Baral, C., editor, *Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering: Papers from the 2006 Spring Symposium*, volume Technical Report SS-06-05, pages 44–49, Menlo Park, CA. AAAI Press.

McGovern, E. A. (2002). *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. PhD thesis, University of Massachusetts Amherst, Amherst, MA.

Moore, M. K. (1975). Object permanence and object identity. Paper presented at the 1975 conference of the Society for Research in Child Development.

Müller, M. (2002). Computer go. *Artificial Intelligence*, 134(1–2):145–179.

Orr, G. B. and Müller, K.-R., editors (1998). *Neural Networks: Tricks of the Trade*. Springer, New York, NY.

Orr, M. J. L. (1996). Introduction to radial basis function networks. Technical report, Centre for Cognitive Science, University of Edinburgh.

Piaget, J. (1954). *The Construction of Reality in the Child*. Basic Books, New York, NY.

Polanyi, M. (1958). *Personal Knowledge: Towards a Post-critical Philosophy*. University of Chicago Press, Chicago, IL.

Poole, D., Mackworth, A., and Goebel, R. (1998). *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, NY.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, Amherst, MA.

Rafols, E., Ring, M. B., Sutton, R. S., and Tanner, B. (2005). Using predictive representations to improve generalization in reinforcement learning. In Kaelbling, L. P. and Saffiotti, A., editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 835–840, Rochester Hills, MI. IJCAI.

Rafols, E. J. (2006). Temporal abstraction in temporal-difference networks. Master's thesis, University of Alberta, Edmonton, AB.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. In Willshaw, D. and Murray, A., editors, *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN 1999)*, pages 569–574. ICANN, IEEE.

Schraudolph, N. N., Dayan, P., and Sejnowski, T. J. (2000). Learning to evaluate go positions via temporal difference methods. Technical Report IDSIA-05-00, IDSIA.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–457.

Shah, P., Schneider, D., Matuszek, C., Kahlert, R. C., Aldag, B., Baxter, D., Cabral, J., Witbrock, M., and Curtis, J. (2006). Automated population of cyc: Extracting information about named-entities from the web. In Sutcliffe, G. and Goebel, R., editors, *Proceedings of the Nineteenth International FLAIRS Conference*, pages 153–158, Menlo Park, CA. AAAI Press.

Shani, G. (November 2004). A survey of model-based and model-free methods for resolving perceptual aliasing. Technical report, Department of Computer Science at the Ben-Gurion University.

Silver, D. (2007). 19x19 go olympiad winner plays Guo Juan, 5 dan professional. Personal communication regarding event from the International Computer Games Association (ICGA) Computer Games Olympiad in The Netherlands.

Silver, D., Sutton, R. S., and Müller, M. (2007). Reinforcement learning of local shape in the game of Go. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1053–1058, Rochester Hills, MI. IJCAI.

Singh, S., Barto, A. G., and Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17: Proceedings of the 2004 Conference (NIPS 2004)*, pages 1281–1288, Cambridge, MA. MIT Press.

Singh, S., James, M. R., and Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In Chickering, M. and Halpern, J., editors, *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, pages 512–519, Arlington, VA. AUAI Press.

Spelke, E. S. (1990). Principles of object perception. *Cognitive Science*, 14(1):29–56.

Spelke, E. S. (1991). Physical knowledge in infancy: Reflections on piaget's theory. In Carey, S. and Gelman, R., editors, *The Epigenesis of Mind: Essays on Biology and Cognition*, pages 133–170. Lawrence Erlbaum Associates, Hillsdale, NJ.

Spelke, E. S. (1994). Initial knowledge: Six suggestions. *Cognition*, 50(1–3):431–445.

Spelke, E. S. and de Walle, G. A. V. (1993). Perceiving and reasoning about objects: Insights from infants. In Eilan, N., McCarthy, R., and Brewer, W., editors, *Spatial Representation*, pages 132–161. Blackwell, Cambridge, MA.

Sutton, R. S. (1992a). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992)*, pages 171–176, Cambridge, MA. MIT Press.

Sutton, R. S. (1992b). Gain adaptation beats least squares? In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pages 161–166.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, R. S., Koop, A., and Silver, D. (2007). On the role of tracking in stationary environments. In Ghahramani, Z., editor, *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML 2007)*, pages 871–878, New York, NY. ACM Press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211.

Sutton, R. S., Rafols, E. J., and Koop, A. (2006). Temporal abstraction in temporal-difference networks. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18: Proceedings of the 2005 Conference (NIPS 2005)*, pages 1313–1320, Cambridge, MA. MIT Press.

Sutton, R. S. and Tanner, B. (2005). Temporal-difference networks. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17: Proceedings of the 2004 Conference (NIPS 2004)*, pages 1377–1384, Cambridge, MA. MIT Press.

Taddeo, M. and Floridi, L. (2005). Solving the symbol grounding problem: a critical review of fifteen years of research. Technical Report 21.05.05, Information Ethics Group, Oxford University and University of Bari.

Tanner, B. (2005). Temporal-difference networks. Master's thesis, University of Alberta, Edmonton, AB.

Tanner, B., Bulitko, V., Koop, A., and Paduraru, C. (2007). Grounding abstractions in predictive state representations. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1077–1082, Rochester Hills, MI. IJCAI.

Tanner, B. and Sutton, R. S. (2005). Temporal-difference networks with history. In Kaelbling, L. P. and Saffiotti, A., editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 865–870, Rochester Hills, MI. IJCAI.

Taylor, M. E., Matuszek, C., Klimt, B., and Witbrock, M. (2007). Autonomous classification of knowledge into an ontology. In Wilson, D. and Sutcliffe, G., editors, *Proceedings of the Twentieth International FLAIRS Conference (FLAIRS 2007)*, pages 140–145, Menlo Park, CA. AAAI Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257–277.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Tsitsiklis, J. N. and van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

van der Steen, J. (2007). Gobase.org - introduction to the elo rating system. `http://gobase.org/studying/articles/elo/`.

van der Werf, E. C. D., van den Herik, H. J., and Uiterwijk, J. W. H. M. (2003). Solving go on small boards. *ICGA Journal*, 26(2):92–107.

Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin and Review*, 9(4):625–636.

Wingate, D., Soni, V., Wolfe, B., and Singh, S. (2007). Relational knowledge with predictive state representations. In Veloso, M. M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2035–2040, Rochester Hills, MI. IJCAI.